

DOCUMENT RESUME

ED 291 340

IR 013 128

AUTHOR Karrer, Urs
TITLE In Search for Quality Enhancement Factors: An Exploration of Production Systems for Quality Courseware Development. Report of an Informative Trip through the U.S.A. in Spring, 1987.
PUB DATE 23 Jul 87
NOTE 101p.; Doctoral Dissertation, University of Zurich.
PUB TYPE Dissertations/Theses - Doctoral Dissertations (041)
-- Reports - Evaluative/Feasibility (142)
EDRS PRICE MF01/PC05 Plus Postage.
DESCRIPTORS Authoring Aids (Programing); *Computer Assisted Instruction; *Courseware; *Instructional Design; Microcomputers; Programing; *Quality Control; *Systems Analysis
IDENTIFIERS Product Development; *Software Engineering; United States

ABSTRACT

Arguing that the production system for computer courseware has a strong impact on quality courseware development, this report offers an introduction into the relationships between instructional system design and software engineering, and investigates courseware quality as a means of identifying quality enhancement factors that can make the difference between high and low quality courseware. Background information covering the focus, approaches, and scope of the study and explaining the goals of the fact gathering trip to selected courseware development centers in the United States is followed by descriptions of 12 courseware production systems, including the overall role of authoring languages and/or systems in courseware development. The development characteristics of production systems--which are the starting point for the establishment of quality enhancement factors--are then condensed from soft data and used to classify two production systems. The quality enhancement factors identified are divided into two groups: (1) necessary factors, which include the integration of instructional knowledge into the development process, the transfer of the design ideas and concepts using an implementation environment that supports the design activities, evaluation of the implementation, commitment of staff involved in the process and use of a team strategy, and the establishment of good communications among staff; and (2) desirable factors, which include working with a programmer who has knowledge of the subject matter and who also has experience. The need for additional research to refine these factors is noted, and more than 50 references and a list of persons interviewed at the various sites are provided. (Author/EW)

* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *

✓ This document has been reproduced as
received from the person or organization
originating it.

□ Minor changes have been made to improve
reproduction quality.

• Points of view or opinions stated in this docu-
ment do not necessarily represent official
OERI position or policy.

Report

IN SEARCH FOR QUALITY

ENHANCEMENT FACTORS:

An Exploration of Production Systems for Quality Courseware Development

Urs Karrer

BEST COPY AVAILABLE

"PERMISSION TO REPRODUCE THIS
MATERIAL HAS BEEN GRANTED BY

Urs Karrer

TO THE EDUCATIONAL RESOURCES
INFORMATION CENTER (ERIC)."

ED291340

IR013128

IN SEARCH FOR QUALITY ENHANCEMENT FACTORS:

AN EXPLORATION OF
PRODUCTION SYSTEMS FOR
QUALITY COURSEWARE DEVELOPMENT

Report of an Informative Trip Through the U.S.A. in Spring 1987

U. Karrer
Tannenweg 14
5703 Seon (AG)
Switzerland

July 23, 1987

Copyright © 1987 U. Karrer

A B S T R A C T

The development of courseware (i.e. instructional software) and its implementation in schools was initiated in the U.S.A. in the late 1950s and early 1960s. Though tremendous work was done at that period of time, interest and funding for courseware development projects were declining due to various reasons. But courseware development is not dead; it got a new impetus from microcomputers and their increasing capabilities. Courseware development seems to be a growing field of interest again.

The starting point of this report is the well-known fact that the overall quality of courseware is bad. A bundle of reasons contribute to and may explain this situation. We support the hypothesis that the production system of courseware has a strong impact on quality courseware development.

A research study was established therefore to investigate the question of (large scale) courseware quality with the aim to identify quality enhancement factors which (can) make a difference between high and low quality courseware in relation to the production system.

The literature review on production systems revealed that it would prove very helpful to study such production systems in reality. An interesting sample of representative production systems was investigated in the U.S.A. in Spring 87.

This report offers an introduction into the relationship(s) between instructional (system) design and software engineering. One of the central findings is that courseware development is a specific task which has to be distinguished from "traditional" instructional material design (e.g. books) and "traditional" software development!

The third chapter of this report describes twelve courseware production systems and outlines the overall role of authoring languages (AL) and/or systems (AS) in courseware development. One of the findings related to AL/AS is that these tools (try to) reduce the programming complexity, but still require the author to provide the instructional design. AL/AS, therefore, can be an appropriate production environment depending on the staff who has to develop which kind of courseware under which circumstances and constraints.

Development characteristics of production systems are condensed from soft data and applied to classify production systems. It is shown that the traditional discussion in this area only covers a small part of characteristics which can be used to describe and classify production systems. Two examples of such production system classifications are presented.

These development characteristics are the starting point for the establishment of (quality) enhancement factors which are identified from soft data. Though some of the factors identified are well accepted (especially in software engineering), others are quite new and not well-known.

The constraints, experiences made, overall situations and the production systems of institutions in this area are very diverse. To conclude, there is not a single and/or unique production strategy and/or approach which could ensure good quality development of courseware alone. But we believe that the (quality) enhancement factors identified (can) discriminate between low and high quality of courseware, e.g. increase the likelihood of quality courseware development, and can be instructional for institutions which recently joined this area, and/or are planning to do so in the foreseeable future.

Finally, some recommendations for further research are made to investigate the area of development characteristics, classifications of production systems, and the quality enhancement factors. A special suggestion focuses on the validation of development characteristics and quality enhancement factors, e.g. through a delphi process.

Two subsequent chapters catalog the literature gathered for further reference and list the names and positions of the people interviewed at the various sites.

Keywords: Courseware, Instructional Software, Computer Based Training, (Quality) Development of Courseware, Production System, Production Strategy, Production Approach, Instructional (Systems) Design, Software Engineering, Software Life Cycle, Carnegie Mellon University, CEEDE, CIDE, Collaboration Model, CONDUIT, CDG at Dartmouth College, ETC at University of California at Irvine, Howard University, University of Texas at Austin, Stanford University, CDP at University of North Carolina at Chapel Hill, Authoring Languages (AL), Authoring Systems (AS), Development Characteristics, Classification(s) of Production Systems, Quality Enhancement Factors

A C K N O W L E D G M E N T S

I consider it an exceptional privilege to have conducted this study. I enjoyed the help of many people and want to express my sincere appreciation. I will long remember all the people I had the opportunity to meet, their investments of time, expertise, and thought. Without their advice and input, the value of this report would be greatly diminished. The people whose names are listed in chapter five are only some of the many who offered their perspectives.

I would especially like to thank all those people in the U.S.A. for their continued support and helpful advice in this study after I returned to Switzerland to establish this report.

Finally I want to thank my doctoral committee - Prof. R. Marty and Prof. M. Greuter (University of Zürich). Their insightful counsel and encouragement to conduct this study and trip to the U.S.A. has made this task more worthwhile.

Please make comments available to:

Urs Karrer
Tannenweg 14
5703 Seon (AG)
Switzerland
Tel: +41-64-55 19 65
E-Mail: Karrer@ifi.unizh.chunet (EAN)

TABLE OF CONTENTS

	Page
1. INTRODUCTION	01
1.1. BACKGROUND OF THE REPORT	01
Renaissance of courseware development?	01
Focus on quality of courseware materials	01
Dissertation focusing on courseware development	02
1.2. STATEMENT OF THE PROBLEM AND TARGETS OF THE TRIP	02
1.3. DEFINITION OF TERMS	03
Courseware	03
Instructional factors	04
(Instructional) effectiveness	04
Quality courseware	04
Production system for courseware development	04
Production strategy	05
Production approach	05
Development characteristics	05
(Quality) enhancement factors	05
Authoring language	05
Authoring system	05
1.4. GENERAL APPROACH	06
1.5. LIMITATIONS OF THE STUDY AND BASIC ASSUMPTIONS	06
Effectiveness of CAI and courseware compared to other instructional media	06
Selection of visited sites	06
Authoring languages and systems	07
Courseware	07
Decisions within a production strategy	07
How data was gathered for this report	07
List of quality enhancement factors instead of a complete model	07
1.6. REMAINDER	08

	Page
2. BACKGROUND: THE RELATIONSHIP(S) BETWEEN INSTRUCTIONAL DESIGN AND SOFTWARE ENGINEERING	09
2.1. INTRODUCTION	09
Instructional Design	09
Software engineering and software life cycle	12
2.2. COURSEWARE DEVELOPMENT: A COMBINATION OF INSTRUCTIONAL SYSTEMS DESIGN AND SOFTWARE ENGINEERING	14
Introduction	14
Comparison of ISD and software life cycle	15
Courseware development seen as integration of software engineering and instructional design	15
2.3. SUMMARY	16
3. PRODUCTION APPROACHES AND STRATEGIES FOR COURSEWARE DEVELOPMENT	17
3.1. INTRODUCTION	17
3.2. AN OVERVIEW OF APPROACHES AND STRATEGIES	17
3.2.1. DESCRIPTION OF PRODUCTION APPROACHES AND STRATEGIES	17
Academia's basic module: A faculty member working with a (student) programmer - a stand alone approach	17
CEEDE: Publishing for grades K through 12	18
The CIDE instructional program development: A customer tailored team approach	19
Carnegie Mellon University: CDEC and CMU Tutor	20
The collaboration model	23
CONDUIT - Towards a service model	25
The Courseware Development Group (CDG) at Dartmouth College	27
ETC/UCI: A team approach for large scale CBL material development	30
Howard University: Faculty driven courseware development	34
The informal small team approach at the University of Texas (Austin)	35
Stanford University: Removing obstacles for (courseware) development	39
The development of a support model at the University of North Carolina at Chapel Hill (UNC-CH)	41

	Page
3.2.2. AUTHORIZING TOOLS/LANGUAGES/SYSTEMS AND THEIR ROLE IN COURSEWARE DEVELOPMENT	44
Trade off discussion	44
Who wants to develop what under which circumstances?	45
Criteria for ideal AS and choosing AS	46
Situation on the marketplace	47
Conclusion	48
3.2.3. SUMMARY AND CONCLUSIONS	48
3.3. IDENTIFICATION OF DEVELOPMENT CHARACTERIS- TICS AND CLASSIFICATION OF DEVELOPMENT AP- PROACHES AND STRATEGIES	49
3.3.1. IDENTIFICATION OF DEVELOPMENT CHARACTERISTICS	49
Introduction	49
Literature review	49
Establishment of a taxonomy of development characteristics	51
Discussion of development characteristics	52
Conclusion	53
3.3.2. CLASSIFICATION(S) OF COURSEWARE DEVELOPMENT APPROACHES AND STRATEGIES	53
Introduction	53
Selection of criteria to classify develop- ment approaches and strategies	53
Two examples of production system clas- sifications	54
Discussion of classification examples	55
Conclusion	56
3.3.3. SUMMARY AND CONCLUSIONS	56
3.4. (QUALITY) ENHANCEMENT FACTORS FOR COURSEWARE DEVELOPMENT (FOCUSING ON THE INTEGRATION OF THE INSTRUCTIONAL KNOW-HOW)	56
3.4.1. IDENTIFICATION OF STRENGTHS AND WEAKNESSES IN COURSEWARE DEVELOPMENT	57
Problems encountered in instructional system development and use	57
Problems encountered in software system development and use	58
A list of ideas, problems and recommen- dations made for courseware development	59
Conclusion	61

	Page
3.4.2. IDENTIFICATION OF (QUALITY) ENHANCEMENT FACTORS FOR COURSEWARE DEVELOPMENT	61
Introduction	61
Review of development characteristic sub- groups	61
Identification of factors	62
Discussion of factors	63
Conclusion	66
3.4.3. SUMMARY AND CONCLUSIONS	66
3.5. SUMMARY	66
4. SUMMARY, CONCLUSIONS AND RECOMMENDATIONS	68
Summary	68
Conclusions and Recommendations	69
5. LIST OF VISITED INSTITUTIONS IN THE U.S.A.	70
6. LIST OF AVAILABLE DOCUMENTS	78
7. REFERENCES	87

1. INTRODUCTION

1.1. BACKGROUND OF THE REPORT

Renaissance of courseware development?

The development of courseware and its implementation in schools was initiated in the U.S.A. in the late 1950s and early 1960s. Universities and some equipment vendors were the leading pioneers at that time. The development of courseware began in Europe - especially in England and Germany - in the 1960s and early 1970s. Though tremendous work was done at that period of time, interest and funding for courseware development projects were declining due to various reasons.

But courseware development is not dead; it got a new impetus from microcomputers and their increasing capabilities (memory, graphics, color, sound). Courseware development seems to be a growing field of interest again. Literature is replete with articles focusing on courseware and its use.¹ On the marketplace, more and more companies are offering courseware or support in development of courseware; more and more authoring languages and systems are emerging.

Focus on quality of courseware materials

But the huge quantity of existing literature and the growing number of courseware and authoring tools does not mean that the overall quality of courseware (production) is good. In contrary, there seems to be an "innovative gap" between theory (all the literature which is published) and the overall quality of courseware which is judged to be bad.

Moreover it is not surprising that many research publications focusing on the effectiveness of courseware material(s) come to different results. We believe that the different findings in many studies (which are structured in different ways) mainly result from using different courseware (which is developed after different production approaches and strategies).

But what could explain the overall bad quality of courseware? Naturally a bundle of reasons contribute to and may explain this situation. We believe that the production system (which includes the production strategy and the production approach) of courseware is one of the (major) reasons to explain the situation. Moreover, we believe that the production system has a strong impact on quality courseware development.

The literature is replete with studies and reports on guidelines in the field of designing instructional materials and courseware, beginning with very simple indications and ending with system design models, and on the potential and effectiveness courseware material(s) offer. This field is so huge that some researchers have begun to investigate time into meta-analyses dealing with the effectiveness of courseware only. Besides there is little literature and publications which focus on the development and the effectiveness of courseware [that is on the application of the instructional design in production systems with the focus on the discrimination between good and bad courseware, that is the effectiveness and quality of the materials].

¹ The 28th ADCIS International Conference Proceedings are a good example reflecting this situation.

Dissertation focusing on courseware development

A research study was established therefore to investigate the question of (large scale) courseware quality with the aim to identify quality enhancement factors for courseware development which (can) make a difference in the development of quality courseware.

The dissertation will focus on two aspects: First on the identification of a basic list of instructional factors (in preparation for the empiric test to conduct later on) and second, the investigation of production systems and the "built in" quality assurance specific to courseware development.

It is the aim of the dissertation to verify these theoretical findings - especially the basic list of instructional factors - in an empiric test in "Mittelschulen" in Switzerland.

1.2. STATEMENT OF THE PROBLEM AND TARGETS OF THE TRIP

To conclude, there can be found a lot of literature about the effects and the effectiveness of courseware materials on one hand and about the design/development of courseware on the other hand, but there is not much literature to find which tries to focus on the integration of the two fields of interest together.

There seems to be a lack of research though there is wide agreement that this question is of strong interest. Bork writes in an article about computer futures for education that ...

... the ultimate test of any method of producing learning units including computer related material is the learning effectiveness of the material produced.² But the present basis of comparison of production systems is weak, coming from very few empirical studies³ of different ways to compose computer-based material.⁴

The same author states:

As yet, we have little experimental evidence about how varying the production system affects the material produced, the costs, or other parameters involved. Experimental studies which produce material with a variety of strategies would be extremely useful. Although many of the components of the production system can be argued on rational, pedagogical, or economical grounds, it will be desirable to have more than arguments available.⁵

² A. Bork, Computer Futures for Education, in: Creative Computing, Vol. 10 (1984), No. 11, p. 179

³ One (limited) example can be found in: R. Garton, M. J. Reed, G. Reed, and E. Stevens, Developing a CBI Course: The Process, in: Proceedings of the 25th International ADCIS Conference, pp. 142 - 148

⁴ A. Bork, Personal Computers for Education, 1985, p. 122

⁵ Bork, Production Systems for CBI, in: Walker & Hess (Eds.), Instructional Software, Principles and Perspectives for Design and Use, Belmont (CA), 1984, pp. 113 - 114

Therefore it became more and more apparent that the work on the analysis and comparison of production systems to determine the instructional factors, deduced from theory, that influence effectiveness, is most important.

Beginning the literature review on production systems (approaches and strategies), it soon became evident that it would prove very helpful - even necessary - to study such production systems in reality. It was found that a good sample of production systems could be evaluated in the U.S.A. as there is much development and many activities going on in courseware development.

With the rationale of the dissertation in mind, we made an informative trip to the U.S.A.* The following targets were set for this trip:

- 1) To visit a fair and good sample of sites which develop courseware;
- 2) To discuss the production systems on the spot with representatives of all the sites visited and to get to know the experience of these pioneers in the field;
- 3) To investigate the question of quality assurance and enhancement factors;
- 4) To try to compare visited sites;
- 5) To evaluate courseware at visited sites;
- 6) To investigate differences between the world of CBT [= computer based training ---> business world] and CBE [= computer based education ---> schools]; and
- 7) To check out trends in the U.S.A. (tools, development procedures, and so on).

Main questions were number one through six; number one through four and number six will be outlined in this report. Number five (and seven) does not make part of this report.

1.3. DEFINITION OF TERMS

For the purpose of this report, the following definitions were used for the terms listed:

Courseware:

In instructional design all the materials which are used in courses are called courseware. Whenever the computer is the delivery device, courseware means to be the software delivered through a computer including (all the) worksheets, -books, (teacher's) manual(s), and so on.

Synonyms often used are educational software, instructional software, CAI, CAL, CBI, CBL, and so forth. We are aware of the fact that some authors make distinctions between some of these acronyms and their application in studies. For the purpose of our study it is not necessary to make any of these distinctions as we do not present any classification of courseware and its implementation in schools.

* From February, 26 to April, 19 I stayed in the U.S. and gathered information. For details on visited sites see chapter five (list of visited institutions).

Instructional factors:

Alessi/ Trollip define instructional factors to be characteristics which affect appearance and effectiveness of courseware, that is the quality of learning.⁷ If the instructional factors are implemented in an appropriate way, the courseware should have the supposed learning effects; if not, the learning effect will be diminished or even be missing.

Synonyms are "basic criteria for the design and production of courseware", "educational and psychological principles and processes", "instructional software design principles" or "instructional characteristics".

Examples of such instructional factors encompass answer judging, feedback organization, text presentation and layout, learner control, identification of targets, use of graphics and color.

For detailed information on instructional factors see the working paper "Ableitung eines Basissets an Wirkungsfaktoren für die Evaluation bzw. empirischen Test und Integration in die Entwicklung von computerunterstützten Lernprogrammen" or the English abstract "List of basic instructional factors for the evaluation and integration in the development of courseware".

(Instructional) effectiveness:

Effectiveness means being able to bring about the result(s) intended. In instruction the results are defined through the instructional objectives and/or targets. An example of such an objective may be: This lesson will teach you to determine whether a painting is of the Classic, Impressionist, or Modern periods.

Quality courseware:

Quality is a very abstract term, needs clarification, and has various aspects which even can stand in antinomial relation(s)⁸ to each other. Some of these are: usability, documentation, efficiency, reliability, flexibility, integrity, consistency, maintainability, testability, robustness, portability, and completeness.⁹

For our concern we define quality courseware to be the result of a production system which develops materials in a predictable manner with small variations in (instructional) effectiveness. One aspect to consider throughout the development of courseware is the integration of instructional factors as guidelines to attain a certain minimum level of (instructional) effectiveness.

Production system for courseware development:

A courseware production system may be defined as the (entire) set of activities beginning with vague ideas about the concepts to be taught and ending with finished materials in the hands of users.¹⁰ A production system is based on a strategy and an approach to develop courseware.

⁷ St. M. Alessi, St. R. Trollip, Computer-Based Instruction, Methods and Development, Englewood Cliffs (NJ), 1985, p. 66

⁸ For further details see: M. Evans, J. Marciniak, Software Quality Assurance & Management, New York, 1987, p. 180

⁹ For details see A. Birolini, Die Qualität von Software, in: Computerworld Schweiz, 1986, No. 17, pp. 7 - 12

¹⁰ A. Bork, in: Walker and Hess (Eds.), 1984, p. 96

Production strategy:

Strategy means the combination of resources to setup and run a production process of courseware development. Questions addressed in this context are decisions about manpower and required skills, hardware and tools used for the development. A well known discussion in this context is the use of authoring languages/systems versus all purpose (high) level languages or the team versus individual approach discussion.

Production approach:

The term production approach encompasses the way and process of production and focuses on the application of the software life cycle and the instructional software design models to courseware development.

Development characteristics:

By development characteristics we mean forming parts and special qualities belonging to production systems which make it possible to classify production approaches and strategies, and guide the identification of quality enhancement factors.

Sometimes these development characteristics are called "procedural guidelines", "procedural informations", or "properties", too.

(Quality) enhancement factors:

Being aware that courseware is a reflection and result of a production system, the question which matters is: Are there aspects and/or factors due to the development characteristics which can make a difference between good and bad courseware, that is increase the likelihood that the effectiveness of courseware produced after these factors will be improved?

We define such factors which can make a difference and may improve courseware quality as (quality) enhancement factors for courseware development.

Authoring language:

An authoring language (AL) is an interpreter which reads commands from a file and executes those commands. Typically, the file is written with a text editor. A small language may have only about a dozen commands; a large language may have two to three hundred commands.

Authoring system:

An authoring system (AS) is a program which automates the writing of a command file or files. It checks for legality of commands, and is usually menu-driven so the author does not have to remember all the commands in a language. It guides the author through the programming process and virtually eliminates the need to know how to program the computer or to know a programming language of any kind.¹¹

¹¹ J. L. Young, The Case for Using Authoring Systems to Develop Courseware, in: Educational Technology, Vol. 24 (1984), No. 10, pp. 26 - 28

1.4. GENERAL APPROACH

Addresses and contacts for the trip were gathered from various sources. A starting point was literature, especially conference proceedings, and Adler's report.¹²

In advance we got in contact with institutions we were interested to visit (by a letter). Arrived in the U.S. by end of February, we established appointments by phone. The periods spent at one site ranged from half a day through three or four days depending on schedules of representatives and wealth of information available and useful for the purpose at hand.

A set of questions was established in advance to conduct the interviews at all the sites. The questions were put together related to a few fields of interest as follows:

- Organization of the institution visited, organizational environment;
- Instructional quality and factors;
- Integration of instructional knowledge into the production system; and
- Future role of computers in education/training.

Additional documents and handouts were gathered where available. Chapter six gives an overview of all the documents which were gathered throughout the trip.

Throughout the trip, it became apparent that it would prove very helpful if a report of this trip would be established. After coming home, we began to outline the report. One important part of this report are the descriptions of representative production systems. These descriptions were sent back to the U.S. for suggestions and checking the adequacy and correctness of the description given. This proved very helpful because it has not been always simple to (objectively) condense the wealth of opinions, information, and handouts, or lack thereof, to reach an understanding of what occurred and why.

1.5. LIMITATIONS OF THE STUDY AND BASIC ASSUMPTIONS

Effectiveness of CAI and courseware compared to other instructional media: Despite many (false) starts and pitfalls by those engaged in applying the computer as a learning aid, we believe that the computer presents a new and unique educational tool yet to be exploited adequately. To make it clear, we believe that the use of courseware as an instructional method can be an efficient enrichment and supplement of existing instructional possibilities and methods.

Selection of visited sites: It was not intended to have a good representation (in a statistical manner) but to visit places of interest for the purpose at hand. In spite of this limitation, we believe that the choice made and the wealth of information gathered gives a good impression, anyway.

¹² K. Adler, The Computerization of Higher Education: The Experience of Pioneers in the Field, ETH Report No. 86-07, Zürich, 1986

This report gives a background of instructional computing at various universities in the U.S.A.

Authoring languages and systems are not our primary concern in this report though we participated at a CBT Conference.¹³ But it was interesting and useful to make the experience that in business authoring languages and systems seem to be an important environment for courseware development - seen at first glance - compared to the schools (especially Higher Education) where a lot of programming is (still) going on in all purpose (high-level) languages. Finally we did not investigate the question empirically why this difference between business and schools (especially Higher Education) seems to exist.

Courseware as term encompasses all instructional materials for schools which are delivered through a computer including (all the) worksheets, -books, (teacher's) manual(s), and so on. Naturally there are various courseware strategies and there are a wide range of taxonomies of courseware. We are especially interested in simulation-based courseware because we believe that simulations (being part of a demonstration, a tutorial, a practice & drill, an instructional game, a test package, and so on) may make good use of the capabilities of the computer as an interactive medium.

Decisions within a production strategy: This report only focuses on the production system related to the production approach and strategy, but not on assembling a team (assessing requirements, skills and competencies of team members), acquisition of hardware and software for development, financing a project, cost of courseware development, and so forth. There is special literature which focuses on these aspects, for instance Marshall-Lippincott.¹⁴

How data was gathered for this report: This report was established based on soft data - discussions, interviews, books, articles, handouts, and so forth.

List of quality enhancement factors instead of a complete model: Though there are some attempts to identify a model¹⁵ - and not only a list of quality factors - to improve quality courseware development, we believe that it is a very complex - even impossible (??) - task to identify a complete model which represents all the variables associated with quality courseware development. We assume that (theoretical) models never include all possible elements of a process, but are (very) selective (and fit only to a particular situation) or abstract (and are not meaningful for a particular application anymore). Therefore we limit our study to identify (quality) enhancement factors which can be applied in (local) production systems which fit the environment best.¹⁶

¹³ It was the 5th CBT Conference & Exposition at Atlanta (GA) that took place March 15 - 18, 1987.

¹⁴ J. Marshall Lippincott, Charting the CBT Learning Curve, The Six Stages of Program Development, in: Training News, Vol. 8, No. 7, pp. 11 - 13.

¹⁵ For instance: T. Dixon, A Suggested Model for Development of Computer Assisted Instruction for Higher Education, ERIC Document No. 250 982, 1984

¹⁶ There are many reasons for this limitation. One relies upon the fact that quality in design and development of courseware and its management is currently still quite variable. Another one relies upon findings of managerial sciences which emphasize that each system is unique due to a specific environment and related circumstances (con-

Even with these limitations, such a study remains an enormous undertaking. What we have produced here is an **approach** to condense as much information as possible from the trip with the target to identify (quality) enhancement factors. It will need **feedback, revision and modification** as we learn more, but it is a **start** that we have found very useful in guiding our own thinking.

1.6. REMAINDER

Chapter two gives an introduction into the background of this report - the relationship(s) between instructional (system) design and software engineering.

Chapter three focuses on production approaches and strategies for courseware development and is the main chapter. It encompasses a description of twelve production systems and outlines the role of authoring tools (authoring languages and systems) in courseware development.

Development characteristics of production systems are condensed from literature review and discussions. These development characteristics are the starting point for the establishment of (quality) enhancement factors which are identified from literature, review of production systems, and discussions.

Chapter four gives a summary, and states conclusions and recommendations.

The remaining chapters present a list of visited institutions in the U.S.A. (chapter five) and available documents gathered in the U.S.A. (chapter six).

2. BACKGROUND: THE RELATIONSHIP(S) BETWEEN INSTRUCTIONAL DESIGN AND SOFTWARE ENGINEERING

2.1. INTRODUCTION

Instructional Design

Instructional design is concerned with understanding, improving, and applying methods of instruction. That is a process of deciding what methods of instruction are best for bringing about desired changes in student knowledge and skills for a specific course content and a specific student population.¹⁷ In other words: Instructional design is a prescriptive process that is essentially concerned with the establishment of principles, methodologies, and techniques that can be used under well-defined conditions to produce effective learning.

Instruction itself must be a set of external events that support the internal process of learning. As instructional theories reflect relationships between teachers' instructional actions and learners' resulting psychological processes, learning theories reflect relationships between learner's actions and their resulting psychological processes.¹⁸

As there are different levels of generality, and the terms "instructional development" and "instructional design" are not used consistently in literature, some authors employ the term instructional development instead of instructional design and vice versa.¹⁹ We will employ the term instructional (system) design for the total set of procedures that are followed in planning, developing, implementing, and evaluating instruction.

The design of instruction was mainly intuitive and subjective prior to the 1960's. As a matter of fact, the quality of instruction developed by such "traditional approaches" varies widely, is usually unpredictable, and is often never determined at all.²⁰ The more systematic model building approach began in the 1950's in response to the wave of programmed instruction! Basically, the perceived need was to design instruction for particular learning tasks, and to have student post-test performance meet pre-specified criteria.²¹

¹⁷ C. M. Reigeluth, Instructional Design: What Is It And Why Is It?, in: C. M. Reigeluth (Ed.), Instructional-Design Theories and Models, An Overview of their Current Status, Hillsdale, 1983, p. 7

¹⁸ L. N. Landa, Descriptive and Prescriptive Theories of Learning and Instruction: An Analysis of Their Relationships and Interactions, in: C. M. Reigeluth (Ed.), Instructional-Design Theories and Models: An Overview of their Current Status, Hillsdale, 1983, p. 65

¹⁹ For details see: R. Taylor, An Analysis of Development and Design Models for Microcomputer-Based Instruction, Diss. [Univ. of Syracuse], 1986, pp. 29 - 30

²⁰ W. H. Hannum, & L. J. Briggs, How does instructional systems design differ from traditional instruction? in: Educational Technology, Vol. 22 (1982), No. 1, pp. 9 - 14

²¹ For a more detailed description see: F. Y. Clarke, An Identification of the Principles and Processes That Are Applicable in the Systematic Design on Instructional Courseware for the Microcomputer, Diss. [Univ. of Houston], Houston-University Park, 1984, pp. 12 - 18 [UMI No. 85-09005]

Ideally the instructional theories should be deducted from learning theories, but the information provided by learning theories is not sufficient (referring to a deductive approach to theory construction). But we suggest that the information provided by learning theories is necessary for building an instructional theory.²²

The total set of procedures that are followed in planning, developing, implementing, and evaluating instruction is called instructional systems design. The procedures are derived from knowledge of human learning relevant for instruction and from the results of empirical data obtained during tryouts of preplanned instruction.²³ To develop instructional materials, special instructional systems design (ISD) model(s) were established from various authors on different and various backgrounds.

Instructional planning encompasses a variety of tasks and procedures. Most important are the task analysis, the selection of media and the establishment of (behavioral and/or cognitive) objectives. Task analysis is a procedure for analyzing the content, type and level of learning involved in an instructional task. The necessity of establishing explicit objectives in observable, measurable terms has been an undisputed cornerstone of instructional design. The selection of appropriate media depends upon the characteristics of the content, the institution(s) in which the instructional material(s) will be delivered, and so on.

Instructional Development is the process of prescribing and using optimal procedures for creating new instructional material(s) in a given situation.

Instructional implementation is the process of prescribing and using optimal procedures for adapting specific instructional material(s) and/or the instructional institution in which that program is being implemented, so as to enable optimal outcomes from that material in that institution. The result of implementation are instructional materials that have been modified in such a way as to result in the optimal effectiveness of that materials.

A most critical step in an instructional system design (ISD) procedure is the evaluation of the product, both during development (formative evaluation) and after completion (summative evaluation). Therefore it is concerned with understanding, improving and applying methods of assessing the effectiveness and efficiency of all of the aforementioned activities. The result of this procedure is a description of weaknesses, consequences, and/or recommendations for improvements.²⁴

²² For details on the problem of deducting instructional theories from learning theories see: C. M. Reigeluth (Ed.), 1983, pp. 3 - 36

²³ D. T. Aronson, L. J. Briggs, Contributions of Gagné and Briggs to a Prescriptive Model of Instruction, in: C. M. Reigeluth (Ed.), 1983, p. 99

²⁴ C. M. Reigeluth (Ed.), 1983, pp. 1 - 36

Romiszowski indicates a list of heuristics helpful for instructional design. in: A. J. Romiszowski, Designing Instructional Systems, Decision making in course planning and curriculum design, 1986, pp. 24 - 29

Andrews and Goodson identified 14 (crucial) steps of instructional design from a meta-analysis of 40 different ISD models:²⁵

a) Needs Assessment:

- 1) Assessment of need, problem identification, occupational analysis, competence, or training requirements;
- 2) Consideration of alternative solutions to instruction;
- 3) Formulation of system and environmental descriptions and identifications, and
- 4) Costing of instructional programs.

b) Task Analysis [WHAT dimension]:

- 5) Formulation of broad goals and detailed subgoals;
- 6) Development of pretest and posttest;
- 7) Analysis of goals and objectives for skills/learning required, and
- 8) Sequencing of goals and objectives to facilitate learning.

c) Learner Analysis [WHO dimension]:

- 9) Characterization of learner population according to age, grade level, prior learning experience, special aptitudes, and disabilities.

d) Means Analysis [HOW dimension]:

- 10) Formulation of instructional strategies to match subject matter and learner requirements;
- 11) Selection of media to implement strategies;
- 12) Development of instructional material based on strategies;
- 13) Empirical tryout of instructional material with learner population, for diagnosis of learning and instructional material(s) failures, and
- 14) Development of materials and procedures for installing, maintaining, and periodically repairing the instructional material(s).

Snelbecker is aware of the problem that instructional theories have not been operationalized in a form that can easily be used by practitioners (individual teachers) in (traditional) instructional system design. On the other hand, designers, theorists, and researchers seem to be more comfortable with the growing number of models and theories. The same author hopes that it is plausible that educators will look to instructional theories and design strategies to produce effective courseware (= instructional software) for the microcomputer. Finally, the application of instructional theory, concepts, and principles shall be much more feasible with a computer-based system than with usual teacher-dominated classroom methods and techniques.²⁶

²⁵ D. H. Andrews & L. A. Goodson, A Comparative Analysis of Models of Instructional Design, in: Journal of Instructional Development, Vol. 3 (1980), No. 4, p. 5 [Cited in: F. Clarke, Diss., 1984, S. 22]

²⁶ This is an interesting opinionion, but the author does not indicate any theoretic or empiric findings which could support this idea. For details see G. E. Snelbecker, Is Instructional Theory Alive and Well?, in: C. M. Reigeluth (Ed.), 1983, pp. 437 - 472

Software engineering and software life cycle

As one of the results of the "software crisis", the idea of software engineering was initiated in the late 1960's. This section focuses on the software life cycle and software quality assurance as important outcomes of this discipline.

Software engineering is an engineering discipline which applies general and/or accepted concepts, methods and principles to software development. In other words: Problems which arise in software development projects shall be solved by engineering methods and principles.²⁷

Apart from the development and presentation of methods²⁸ and the implementation of software engineering environment(s) (tools), the software life cycle is seen as one of the most famous and acknowledged results of the software engineering discipline for the time being.

The concept of the software life cycle can be seen as a generic term encompassing the complete software development process. This process can be broken down into several phases as follows:²⁹

- a) Requirement specification or system definition phase,
- b) Design phase,
- c) Coding/Implementation phase,
- d) Test/Evaluation phase, and
- e) Maintenance phase

The software system definition phase is usually the most complex and important phase of software development. It is here that initial requirements are set down, and the system and software project should be structured and defined. Conducting this phase is crucial for software development.

The (top and detail level) design phase is the first step in the transition of requirements to a physical system. Requirements are allocated to individual parts of the software system to take shape (top down modularization). Depending on the complexity and the size of the software system to develop, this phase (of top and detail level) design may proceed through a number of subordinate levels. The required result of this phase is the completion of the total design in preparation for the coding/implementation.

²⁷ Examples of state of the art descriptions can be found in:

- a) W. Hesse, Methoden und Werkzeuge zur Software-Entwicklung - Ein Marsch durch die Technologie-Landschaft, in: Informatik-Spektrum, Vol. 4. (1981), pp. 229 - 245
- b) P. Hruschka, Integrated system engineering environments, in: Elektronische Rechenanlagen, Vol. 27 (1985), No. 2, pp. 60 - 68
- c) R. Tobiasch, Tendenzen in der Softwareentwicklung, in: Technische Rundschau, 1986, No. 52, pp. 50 - 57

²⁸ I.e. SADT, structured design, Jackson systems design, entity-relation approach, and so forth.

²⁹ Shneiderman describes the life cycle concept as a framework that can be adapted to meet the widely varying needs of specific projects in: Designing the User Interface, Strategies for Effective Human-Computer Interaction, Reading (MA), 1986, pp. 413 - 416

There are various approaches to conduct the coding phase. It is up to the project manager and/or team members (programmers) to make up their mind to choose a coding approach. This phase should take place in a planned manner, whether a bottom up or a top down approach to implement/code modules is applied.

After coding the implemented software system has to be tested and reviewed for fulfillment of required content, performance, and so on. Here again, it is up to the responsible team members to decide on an approach for the testing phase.

No matter what the development or implementation approach, after each unit is reviewed (and released) it is placed under configuration control in the software development library. This is a condition for a well organized and planned maintenance phase which can continue for years.

The life-cycle presented above follows an orderly step-by-step manner, but the reality is often iterative, requiring a return to earlier stages for some parts of the system design.³⁰ The result of this model is a series of parallel miniprojects, each building on an incomplete data set, providing a specific piece or category of data to other miniprojects and requiring data from prior or concurrent parallel phases (interface definition!!). Moreover the software life cycle does not cover project organization, management activities going along with software development,³¹ and latest methods as "rapid prototyping". Nevertheless the establishment of the life cycle concept can be seen as an important step and success of software engineering.

It was mentioned above that it is important to test and review an implemented software system for fulfillment of content requirements, performance, and so on. But this is not enough to arrive at a quality level required by the system definition phase. Therefore quality must be built in into the software system from the very beginning - by establishing an effective development environment. Quality assurance can be defined as the total effort by all concerned with the software development to ensure that a quality software system is developed. One of the problems now is that the measurement of software quality is not yet a science and will not be until effective means are developed, accepted, and implemented to qualify the software system developed. Moreover there are various concepts, activities or management methodologies that have been established and applied to quality assurance in software system development.³²

³⁰ For further details see: M. Evans, J. Marciniak, 1987, pp. 22 - 26

³¹ An overview on project management methods can be found in: S. K. Gupta, L. R. Taube, A State of the Art Survey of Research on Project Management (1976 - 1983), in: Burton V. Dean (Ed.), Project Management: Methods and Studies, Amsterdam/New York/Oxford, 1985, pp. 293 - 313

Hughes, et. al present a team approach for courseware project management, in: Technical Management of the Development of Computer-Assisted Instruction for Complex Content, in: Performance and Instruction Journal, 1986, August, pp. 19 - 24

³² For further explanations on such methodologies see: M. Evans, J. Marciniak, 1987, pp. 5 - 8

As stated in the introductory chapter, the ultimate test of any method of developing software system(s) is the quality of the software produced - transferred to courseware development the learning effectiveness of the material produced. We assume and support the approach that the quality (of the software system developed) is a result of the application of sound engineering practice. And moreover, it is the project infrastructure (= production system) which provides the framework through which software quality is "built in" the software system.³³ Evans and Marciniak state that there has been a continuing awareness of critical problems encountered in developing systems involving software. But there has not been a corresponding awareness of the effect that the development process has on product quality and the effects that result from poor integration of the individual process into an effective development infrastructure.³⁴

This introductory part focused on the concepts of instructional system design (ISD), software engineering, software life cycle, and questions of quality assurance of software development. The next part will introduce the relationship between the ISD process and the software life cycle in conjunction to courseware development.

2.2. COURSEWARE DEVELOPMENT: A COMBINATION OF INSTRUCTIONAL SYSTEMS DESIGN AND SOFTWARE ENGINEERING

Introduction

As shown in the preceding part of this report, the total set of procedures that are followed in planning, developing, implementing, and evaluating instruction is called instructional systems design. The procedures are derived from knowledge of human learning relevant for instruction and from the results of empirical data obtained during tryouts of preplanned instruction. To develop instructional materials, special instructional systems design (ISD) model(s) have been established from various authors on different and various backgrounds.

On the other hand, the concept of the software life cycle can be seen as a methodology and technique within software engineering and encompasses the complete software development process. This process can be broken down into several phases as follows: System definition phase, design phase, coding/implementation phase, test/evaluation phase, and the maintenance phase.

³³ M. Evans and J. Marciniak support this approach in their latest publication, too. [See: Software Quality Assurance & Management, New York, 1987, p. 6 and 142.]

³⁴ W. Evans, J. Marciniak, 1987, p. 67

Comparison of ISD and software life cycle

The following illustration compares the overall phases within ISD and the software life cycle (on a condensed and simplified level). This comparison reveals that the two domains ISD and software life cycle have many concepts and ideas in common!

ISD concept	software life cycle concept
Planning phase [including task analysis, statement of objectives]	System definition and design phase [requirements are set; requirements are allocated - the design takes place]
Development phase [creation of the instructional material(s)]	Coding and implementation phase
Implementation phase [adaptation to institutions in which material(s) will be used]	Testing [check and review of software system] ... and ...
Evaluation phase [assessment of the effectiveness and efficiency of the material(s)]	... evaluation phase [assessment of the quality of the software system] Maintenance phase

Moreover this illustration reflects the idea that instructional system design (ISD) and the software life cycle are two domains with strong similarities. As the ISD was/is used for developing "traditional" instructional material(s), the software life cycle is applied for the development of software systems.

Courseware development seen as integration of software engineering and instructional design

Courseware development is a reflection and result of both - instructional system(s) design (ISD) and the software life cycle. On one hand, courseware represents a kind of instructional material which has to fulfill requirements imposed by instructional needs and taken care of by the instructional system design process. On the other hand, courseware represents a software system that is delivered through a computer and therefore should fulfill requirements imposed from software engineering and the life cycle concept.

This finding shows an important consequence for courseware development to consider in a production system:

This ambiguity of courseware development requires that production systems (project infrastructure) are established **specific** to courseware development. A production system for courseware development has to **integrate** ideas of **instructional** design, therefore, and respect the preliminaries of **software engineering** on the other hand.

This makes courseware development a **specific** task which has to be distinguished from "traditional" instructional material design (e.g. books) and "traditional" software development!

2.3. SUMMARY

This chapter has given an introduction into the background of this report: the relationship(s) between **instructional systems design** (ISD) and **software engineering**. The overall and elementary procedures of ISD and the software life cycle have been introduced. A special point was made on the importance of **quality assurance** which can be defined as the total effort by all concerned with the software development to ensure that a quality software system is developed. One of the problems now is that the **measurement** of software quality is **not yet** a science and will not be until effective means are developed, accepted, and implemented to qualify the software system developed.

Secondly, the finding was presented that courseware development is a **combination** of **instructional systems design** and the **software life cycle** concept; that is **each** procedure (on a condensed and simplified level) of the ISD domain and the life cycle concept has its counterpart! This was revealed based on a **comparison** of the overall phases within ISD and the software life cycle (on a condensed and simplified level).

Finally, it was shown that courseware development is a reflection and result of **both** - **instructional system(s) design** (ISD) and the **software life cycle**. This **ambiguity** of courseware development requires that production systems are established **specific** to courseware development and make it a **specific** task which has to be distinguished from "traditional" instructional material design and "traditional" software development.

3. PRODUCTION APPROACHES AND STRATEGIES FOR COURSEWARE DEVELOPMENT

3.1. INTRODUCTION

It has been hypothesized in the first chapter of this report that the courseware production system itself is an important factor to ensure quality courseware and that the design of interactive courseware material is not a well-developed engineering discipline.

In this chapter an overview of such production systems (approaches and strategies) is given, development characteristics are discussed (in conjunction with two example classifications of approaches and strategies), and (quality) enhancement factors for quality courseware development are identified.

3.2. AN OVERVIEW OF APPROACHES AND STRATEGIES

3.2.1. DESCRIPTION OF COURSEWARE PRODUCTION APPROACHES AND STRATEGIES

Literature is replete with information focusing on the design, instructional factors and instructional methodologies and the effectiveness of courseware. On the other hand there is little literature and publications which focus on production systems itself with a strong commitment to quality courseware development.

The following overview outlines the main aspects and features of twelve production approaches and strategies (in alphabetic sequence) which can be found in institutions in the U.S.A. and/or are described in the literature.

Academia's basic module: A faculty member working with a (student) programmer - a stand alone approach

The starting point in many academic places was/is the team work between a faculty member who is working together with a (student) programmer. They work on their own (without any help from a support/coordination group). We call this production strategy a "stand alone approach".

The faculty member is the "spiritus rector" of the whole project. Consequently, a representative of this strategy distinguishes between "traditional software engineering projects" and instructional software projects. In his view the traditional (recommended) strategy is team-based and works with guidelines, standards, documentation, and forms. On the other hand, the instructional software project is seen in a three phase model:

- 1) Have one creative idea;
- 2) Then sit together for some weeks of brainstorming (as a sort of outlining and designing); and
- 3) Finally a process of 1 - 3 years of iteration (coding, revision, fine tuning, evaluation, and so forth) is needed.

Typically this production approach is driven by the faculty member's heart and mind commitment far away from the traditional "bureaucratic" approach. The idea (at the very beginning) and its implementation is the very heart of attention. This requires a willingness to throw things away and to go through a process of iteration which takes a long time (years).

The brainstorming normally takes place together with the programmer. The motto for these weeks is "Go as far as possible - be creative".

The faculty member and the programmer are working in an atmosphere which may be described as "highly motivated" and "open minded" in the sense of knowledge exchange between the faculty member and the (student) programmer (about computers and programming on one hand and about content and/or pedagogical aspects on the other hand).

A critical constraint of this approach and strategy is the relationship between the faculty member and the programmer: If the relation is bad-tempered, the project will not make progress.

CEEDE: Publishing for grades K through 12

The Center for Educational Experimentation, Development and Evaluation (CEEDE) is part of the University of Iowa at Iowa City (IA) and develops resp. publishes courseware and other instructional (print and audiovisual) materials for grades K through 12 all over the world.

CEEDE is supported through sales of materials, federal funds and grants of various sources.

CEEDE is a nonprofit organization which started in 1977 on PDP 11 machines, changed to personal computers and made a commitment for Apple II+, IIe, or IIC machines. CEEDE disks for the Apple require only 48K RAM, Dos. 3.3 and one stand alone floppy drive. Disks are also available which are designed to run on a local area network consisting of Apple computers, a Corvus hard disk drive, and networking hardware and software.

Its materials are designed for limited English proficiency (LEP) students, as well as for use in English as second language (ESL) classrooms.

All materials are available in English, and many are also available in home language versions. Many materials are available in Cambodian, English, Lao, Hmong, Spanish, and Vietnamese.

The subject matter areas are languages, arts, mathematics, social studies, science, health, and practical life skill/career education.

It is CEEDE's policy to try to facilitate the transition of instructional materials into "traditional" (regular) classrooms: The materials are developed using objectives from the regular curriculum. For instance, the reading difficulty levels for the English language materials have been established and verified

using the Flesch and Wheeler readability formulas. The Spaulding formula was used to determine the readability levels of Spanish language materials.

All the courseware is coded in BASIC. CEEDE developed a wide range of "drivers" - a sort of tools for handling the man-machine interface (input, output, appearance of screens). This library of drivers is used heavily for the development of new instructional software.

Actually it is not planned to work with any Authoring Language or System, but it is possible that there could be a change from BASIC to PASCAL.

The general development process can be characterized as follows:

- Find funds, grants and/or somebody who wants to have produced a certain instructional material package;
- Proposal phase and decision making phase;
- Design phase;
- Implementation at CEEDE (by using the drivers); and
- Distribution phase.

CEEDE made good experiences with the drivers aforementioned for the courseware development. This sort of library of modules is very important and allows the developers to have consistent interfaces over a wide range of courseware materials.

To conclude, CEEDE as a non-profit organization develops and publishes instructional material all over the world for grades K through 12 and focuses its work on limited English proficiency and English as a second language besides other curriculum domains. The (library of) drivers (for Apple II machines) are crucial for CEEDE's courseware development. The materials are developed using objectives from the regular curriculum; this reflects CEEDE's policy to try to facilitate the transition of instructional materials into "traditional" (regular) classrooms.

The CIDE instructional program development: A customer tailored team approach

As part of The University of Maryland University College, the Center for Instructional Development and Evaluation (CIDE) at College Park (MD) is in the business of designing, developing, and evaluating instructional materials since about five years. Its services range from the design of traditional programs to the development of advanced CBI using interactive video to build awareness and skills and research to the evaluation of educational systems, materials, and equipment.

One of its first projects was a learning system to teach basic educational skills to U.S. soldiers. The involvement in PLATO as a subcontractor was another important milestone in

CIDE's own history. CIDE works for customers exterior to the university.

Over the five years CIDE's process of developing instructional materials has changed. At the beginning the materials did not achieve a quality level as it should be. In addition CIDE was confronted with deadline and budget problems. Through the PLATO/Control Data project CIDE learned to ensure quality.

Nowadays CIDE employs the ISD process. After CIDE's representatives ISD provides a careful, systematic approach to meeting the complex instructional needs of a project.

CIDE's team strategy is group centered. That means that all persons involved in the development are following the steps of the ISD process together. This is valid for the design phase especially.

All these features and the close contacts with its customers within a project give birth to a tailored approach which results with original solutions to specific educational needs.

CIDE begins a project with a comprehensive needs assessment. Through interviews, on-site inspections, and research, qualified CIDE staff analyze the task and design the material(s). At all stages of development, the material(s) is/are evaluated and revised, incorporating feedback from the educational system. This evaluation and revision is ensured by special quality control staff. That means that staff outside a project makes the reviews and evaluations.

To conclude: CIDE's approach developed over time with projects. Its important features are the ISD process, a group centered team strategy due to the fact that CIDE is in the communication business, iterative evaluation and revision administered by special quality control staff. This gives birth to a customer tailored approach which results are quality materials and systems.

Carnegie Mellon University: CDEC and CMU Tutor

CMU's goal related to educational computing is to become a leader and model among universities by making a commitment to pioneering environments for academia. To realize this goal, a mixture of individual, departmental and central actions have been initialized.³⁵ One of these actions was the establishment of the Center for Design of Educational Computing (CDEC). The goal of CDEC is to provide a productive working environment for faculty who are developing educational software.

To meet this goal, the CDEC performs four major functions:

- Introducing software developers with similar or complementary interests (Colleagues for the educational software developer come from experts in various disciplines and from other de-

³⁵ For details see: K. Adler, 1986, pp. 25 - 38

An overview is given in: D. Osgood, The Difference in Higher Education, in: Byte, 2/1987, pp. 165 - 178

velopers addressing similar intellectual problems);

- Increasing access to developers' information sources (through bi-weekly seminar series on campus, newsletter, educational software library, and so forth);
- Establishing means to maximize dissemination of high-quality instructional software (consulting function, commercialization of instructional software through an arrangement with McGraw-Hill, Inc. and non-commercial distribution through university consortia); and
- Helping to provide direct development support (help faculty obtain funding, sharing of CDEC staff programmer experienced in design of instructional software, and so forth).

At the beginning of the Andrew³⁶ project, some faculty members have designed courseware, had it implemented in C, and used it in their courses. The faculty members worked together with a (student) programmer.³⁷ The problem arrived at that time that the programmer graduated (and left the university) but the faculty member found the program needed to be revised and updated!

While the system software at CMU is written in C, this language was to be found not well suited for applications programming by non-expert programmers. To give such people the ability to create complex interactive graphics programs which exploit the rich capabilities of the workstation³⁸ environment, Bruce and Judith Sherwood have created an integrated programming environment called CMU Tutor.

While patterned on the MicroTutor language developed in the PLATO project at the University of Illinois, CMU Tutor has enhancements for the workstation environment, including pop-up menus, variable window size(s), mouse support, etc. By keeping track of which routines have been changed and therefore need recompilation, CMU Tutor operates as an incremental compiler, which combines the revision speed of an interpreted language with the execution speed of a compiled language. A tightly integrated graphics editor generates source code from a display created using the mouse. An on-line help facility not only describes the language features in detail but provides sample routines which can be run immediately thanks to the multi-window and incremental-compiler environment.³⁹

³⁶ Andrew is the computing environment under development by the Information Technology Center (ITC), a joint project of CMU and IBM. Andrew runs on a campus wide network of high function workstations. The two major components of Andrew are a file system and a user interface.

³⁷ See the description of "Academia's basic module: A faculty member working with (student) programmer".

³⁸ These machines are powerful, 32-bit processor workstations, with features including virtual memory and bit-mapped graphics display.

³⁹ B. A. Sherwood, Workstations at Carnegie Mellon, IEEE Computer Society Reprint, November 1986, p. 16

In a series of funded workshops (during the summer of 1986), educators from around the country (ICEC)⁴⁰ got a chance to use CMU Tutor and assess its value for their own institutions.

CMU Tutor has been available for about a year and is still under development. CMU Tutor programs run on Andrew workstations and on popular micros, including the IBM PC and the Macintosh. Although the workstation environment is a more productive one for the creation of new instructional software, limited authoring capability is available on the Macintosh. By Summer 87, a plateau in the development of CMU Tutor will be reached.

The experience of CDEC shows that CMU Tutor is useful for both experienced programmers and those with limited programming experience. Developers can create interactive applications, especially educational software, for workstations without having to depend completely to the assistance of professional programmers.

CMU Tutor as an authoring language (AL) can drive the development process towards the individual strategy. This does not mean that this is the only possible approach: Different people do it in different ways and the CDEC staff can support faculty developers.

In the handout *Computer-Based Tutor on Graphic Equations* the authors describe the production approach they applied to design and implement the program *Sketch*.⁴¹

The development process was split up in

- A preliminary design session (Two top mathematics and science teachers from local public schools at the secondary level were selected, involved in brainstorming);
- A first prototype generation phase (Graphical displays were generated with CMU Tutor and used to investigate how the teachers' content idea could be presented.);
- Subsequent design sessions (detailed discussions of the interactions that might take place as a student proceeded through several sample problems);
- The program development (The design sheets were the base for the generation of a general program);
- A field testing phase (split up in an informal test at CMU and a more systematic test conducted in the Fall of 1986, when the *Sketch* was made available to students at a local high school for two days); and
- A revision phase (It was possible to make some small but crucial changes during the five minute break between class periods at high school and to use the revised program with students immediately!).

The issue of quality courseware development is addressed in various ways; most important is the generation of a sort of prototype with CMU Tutor, which is started through the design of displays which could be used in the final program.

⁴⁰ ICEC = Interuniversity Consortium for Educational Computing

⁴¹ For details see: D. Trowbridge, J. Larkin, C. Scheftic, *Computer-Based Tutor on Graphic Equations*, CDEC, Pittsburgh (PA), 1987

In summary, CMU's goal related to educational computing is implemented by a mixture of individual, departmental and central actions. One of these actions was the establishment of the Center for Design of Educational Computing (CDEC), whose goal it is to provide a productive working environment for faculty who are developing educational software.

The fundamental development philosophy is: "Different people work in different ways". From the point where a faculty member has the hardware, Andrew and CMU Tutor, he can be alone but CDEC may support him in his activities and/or assign a (student) programmer to help him coding in CMU Tutor

The collaboration model

Collis and Gore⁴² describe a model for software development, called the "collaborative model". This model is implemented through a seven step phase procedure and was tested through various projects in the university environment. Special emphasis is given to the documentation that supports each of the phases.

The collaborative model is based on three sets of principles. They are:

- Principles reflecting the needs of the software engineer as:
 - 1) Use documentation as a software design medium.
 - 2) Write software requirements that are as complete as possible.
 - 3) Structure the development process so that distinct times are provided for planning and evaluation prior to implementation.
 - 4) Partition software projects into components that can be developed and tested independently.
 - 5) Design systems so that they are easily contracted or extended.
- Instructional design principles are:
 - 1) Identify intentions of the material relative to learning objectives, target learners and overall instructional strategy.
 - 2) Delimit the scope and sequence of the instructional content.
 - 3) Specify the methodological decisions involved in delivering the content.
 - 4) Establish evaluative processes for both the learner and the designer/instructor.
- Collaboration principles are:
 - 1) Design through iteration and compromise.
 - 2) Assess congruency at specific point in the development process.

⁴² B. Collis and M. Gore are members of the IBM-UVic Project at Univ. of Victoria, Victoria (BC) in Canada.

My description of the collaborative model is based on the following article: B. Collis and M. Gore, The Collaborative Model for Instructional Software Development, in: Educational Technology, February, 1987, pp. 40 - 44

- 3) Minimize the common knowledge necessary for all participants and document this shared knowledge in a "collaborative filter"⁴³

An iterative seven step phase model implements the process of design through documentation as follows:

	Intents ⁴⁴	Contents ⁴⁵	Method	Evaluation	Computer Scientist	Program
Phase 1	1 st draft					
Phase 2	2 nd draft	1 st draft C. Decision Path 1 st draft ⁴⁶				
Phase 3	Final draft	2 nd draft C.	Design Manual(1 st draft)			
Phase 4 ⁴⁷		Final draft C. Dec. P. 2 nd draft	Utility Tools Design Manual (2 nd) ⁴⁸		Requirements document (1 st)	
Phase 5		Dec. P. finished	Design M. (complete) Teachers Mat (1 st)	Plan program evaluation strategy	Requirements doc. (2 nd)	Prototype
Phase 6			Teacher's Materials	Implement + assess program evaluation	Requirements document (near complete)	Usable Program
Phase 7			Teacher's Materials finished	Evaluation report finished	Requirements doc. finished	Program finished

⁴³ The most important part of developing instructional software involves the process of design through documentation. The authors call the documents that are needed for collaboration "filter documents".

⁴⁴ "Intents" involves defining the purpose for which the courseware is being developed.

⁴⁵ "Contents" refers to the selection and sequencing of the specific contents and data to be included in the software.

⁴⁶ The decision path chart illustrates the decisions made concerning the sequencing of the contents of the program.

⁴⁷ Phase 4 includes congruency checks.

⁴⁸ It is with the design manual that the educator (SME) must begin to define how the program is to appear to the outside world. It will eventually function as a "storyboard" of all possible screen displays as well as the documentation to describe all legal and illegal input.

To conclude: A critical feature of the collaborative model is to be said that the actual coding of courseware does not take place until all decisions involved in the four instructional design components are well documented. The authors believe that it emphasizes the nature of "filter documents" whose aim is to combine maximal information exchange with minimal paperwork and minimal interaction between the various participants in a software development.

CONDUIT - Towards a service model

CONDUIT is a nonprofit organization that develops and publishes educational computer software for high schools and colleges. Organized in 1971 with funds from the National Science Foundation (NSF), CONDUIT is currently affiliated with the University of Iowa at Iowa City. Its primary purpose is to improve learning through the use of computer technology. CONDUIT seeks to distribute quality materials directly to schools and departments, as well as through various distributors (such as catalogue houses and computer stores) across the country.

Its aim is to provide instructors with a reliable source for ideas and software and to assist authors in developing useful computer-based packages. CONDUIT is supported primarily through the sales of materials, supplemented by federal grants, membership fees, and vendor contracts.

CONDUIT relies primarily on submissions from authors, typically classroom instructors who have already developed materials for their students, for its source of packages to publish. CONDUIT has developed a variety of guidelines, tools, and standards to assist these authors with instructional design, programming, and documentation. Consequently, an (outside) author is in charge of the instructional software, but CONDUIT has the resources and services to ensure good quality production of these materials.

To ensure its materials meet high standards of quality, these submissions undergo CONDUIT's rigorous selection process before being accepted for publication. An important step in this process is peer review by leading educators for conceptual validity and instructional usefulness. CONDUIT's staff also reviews submissions from instructional design and technical aspects.

CONDUIT has three major divisions in addition to an advisory board. Research and development looks at new machines and software and develops tools for authors. Product teams work with authors, series editors and reviewers in preparing new products. These teams oversee products in the following disciplines: biology, chemistry, economics, education, English, languages, mathematics, music, physics, political science, psychology, sociology and statistics. The other division is marketing and distribution, which interacts with customers from around the world, but primarily in English-speaking areas.

The tasks in working with a new product can be briefly described as follows:

- Candidate identification: supported by submission guide and author development notes;
- Candidate review: supported by peer review form and series editor summary assessment form;
- Publication approval: conducted through product analysis and market surveys;
- Contract: author agreement
- Revision: working with the author to revise the software and printed documentation to meet CONDUIT's guidelines and review suggestions;
- Testing: alpha testing inhouse by another product team (no formal guidelines -- based on experience) and beta testing with target audience (beta test form is being developed);
- Manufacturing: manuals are printed and disks are duplicated off-site, then packages are assembled; and
- Announcement: package is promoted through CONDUIT's catalog, fliers, conference presentations and exhibits, and magazine and journal advertisements.

CONDUIT's involvement in educational software development with authors (= content experts) can be described as a **service model**. It provides or locates the necessary skills to complement the authors' abilities. These include programming, instructional design, and editorial skills. CONDUIT would like to evolve this informal mode of operation into a **development center** for faculty where people with certain skills (SME) meet people with other skills (instructional designer, programmer, publishing, and so forth).

CONDUIT supports author design and development through printed guidelines and standards and through software tools. The tools have been of limited success, probably due to the programming skills needed to use them effectively. Some of the tools are machine-code extensions to the BASIC programming language and others are a development shell of BASIC. CONDUIT is currently creating a library of C language routines that will be useful to educational software authors.

One of the best tools that CONDUIT has recently used for development is a **prototyping tool**. It is a good tool for a team to use in CONDUIT's opinion because it allows the group to quickly develop an interactive model of the intended product. This explicit model is easier to critique by project members because it is expressed in the intended medium (= the computer). Structure, flow of control, and interaction can all be expressed better than in a static form like storyboards. Another advantage is that project members can modify the model relatively quickly to test "what if" questions.

Another concern of CONDUIT's development concerns is the prominence of the evaluation phase. CONDUIT prefers an **iterative development model** that cycles typically three times through design, develop, evaluate, and revise phases. This approach encourages more testing and revision based on testing. It also minimizes the losses of a poor or incomplete design.

The Courseware Development Group (CDG) at Dartmouth College

The goal of the microcomputing project at Dartmouth College is to explore and develop the possible uses of the computer as a tool to improve the educational process. The Macintosh was chosen by the College to be strongly recommended. One aspect of the microcomputing project activities is the development of instructional materials - discipline specific courseware and generic authoring tools.

Computing Services at Dartmouth College has instituted a Courseware Development Group (CDG),⁴⁹ whose claim is to work with faculty to develop quality courseware. This approach was selected because the Macintosh was judged to be too complex to handle (too time consuming) compared to the earlier mainframe environment where faculty members worked closely in conjunction with students to develop applications (see academia's basic module).⁵⁰

The educational software projects are financed by the college (50 %) and through grants from foundations (50 %) Dartmouth received in 1983 and 1985 to develop educational software on personal computers in the humanities and in science. With a focus on pedagogical excellence and quality, teams of trained designer/programmers work together with interested faculty to develop instructional materials (specific courseware and generic authoring tools which should facilitate faculty in writing their own materials).

The Courseware Development Group (CDG) develops courseware for diverse disciplines ranging from music up to the sciences. All materials are available through Kinko's.

The production approach emerged in two phases: A starting phase beginning 1984 with projects whose nature was very much exploratory in content and procedures and the approach representing the actual phase 2. The CDG finds this actual approach to be a fruitful technique in collaboration between an instructor, usually a faculty member, and the CDG developer(s). In DCG staff's opinion this style makes good use of the college's limited people and monetary resources.

The development process can be split up in a:⁵¹

- Proposal phase;
- Initial design phase: group meeting(s);
- Phase of detailed design,
- Coding phase, and a
- Formal evaluation and revision phase.

⁴⁹ The group had six designers/programmers at the beginning of the year but the staff actually changes.

⁵⁰ The programming language BASIC was developed with this production strategy of team work between faculty member(s) and students.

⁵¹ The following description can be found in: St. Maker, The Courseware Group at Dartmouth College, in: SIGCUE Outlook: Computing in Higher Education, Vol. 19 (1986), Nos. 1/2, pp. 20 - 24

Proposals are selected for pedagogic value, technical feasibility, and for estimated impact. The CDG is looking for programs of high generality across fields, or of tight focus within a field.

Faculty is stimulated by showing them existing Mac programs and discussing possible programs. The CDG seeks committed teachers, with a "Let's do it" attitude".

The CDG as a whole works with the faculty member in defining needs and in overall design of the application (= initial design phase), especially full use of the Macintosh user interface, in several brainstorming sessions.

It is an advantage for the overall process having designers/developers and faculty member(s) together in this phase: Mutual idea generation occurs which may originate ideas which carry the faculty member's concept far beyond the original proposal. It is the faculty member's task to keep the concept pedagogically sound and focused on usefulness in courses.

The detailed design is done by the CDG. Its results are the screen design and a feature description. A screen layout design aid in MacPaint is used to outline possible screens. The screen design and the feature description is discussed with the faculty member to get an agreement on the further development.

A designer/programmer is assigned as the developer to work closely with the faculty member on the project, having meetings together as appropriate. The coding is totally done by the CDG. Whenever a "checkpoint" is reached, the faculty member has to give comments and evaluate the actual "prototype". Heavy attention is on the pedagogical soundness (to ensure by faculty member) and a good user interface. All the coding is done in **PASCAL**.

In later stages, testing is done with potential students, or in an actual class (formal evaluation and review phase). It is possible that two to three revision phases have to be done until it is accepted to be pedagogically sound and has an adequate user interface.

At phase 1 of the courseware development project was the idea that everybody would have all the ideas of the design phase(s) in mind. The results were problems and misunderstandings in the ongoing development afterwards. Nowadays the process is more document-based. The feature description is a sort of time-checklist in which possible features are mentioned and their priorities are rated high and/or low. Depending on time and money, more or less of the (lower priority) features will be implemented, too. After the CDG's opinion, this priority list is a good mean to adapt to the situation that they often do not exactly know at the (very) beginning (in the initial design phase) where they will go, especially if there are no similar projects to compare.

The CDG made discoveries and identified problems about the process of developing educational software at Dartmouth:

- related to the project in general:

- 1) It is important to share ideas. Therefore the CDG tries to work with faculty closely.

The point at first is to generate new ways of thinking about the concept and problem at hand. The CDG does the user interface design as a group together with faculty to help everyone get their minds open minded: Brainstorm, innovate, bounce around ideas is the motto.

- 2) Factors which increase project cost(s) are inexperienced, slow, or just plain incompetent designer/programmers (probably the major factor); working on an authoring tool versus a smaller program; poor project definition or lack of faculty commitment; complex user interaction; the use of more than one programmer sequentially.
- related to designers/programmers:
 - 3) If they are to manage themselves at all successfully, they should have significant experience outside courseware on large, self-managed projects, with a modern all purpose (high level) language (HLL).
 - 4) Teaching new programmers by reading programs seems to fail; they must build programs themselves, in order to actually cope with the problems at first hand.
 - related to project management/communication:
 - 5) Use design documents and sample screens, but keep it quite informal.
 - 6) Design checkpoints into the implementation schedule for a project, so that work can be redirected easily. At a checkpoint, the program has to be robust, clean, documented, and usable, but it may lack desired features (see description of feature list above).
 - related to the program:
 - 7) If a program is simple enough to use that only on-line help is necessary, there has been a great saving in not writing a manual. There should always be on-line help.

The overall major problems in courseware development were described as follows:

- The expense and time of development;
- The difficulty of prototyping quickly;
- The generally weak programming environments available; and the problem that
- The CDG has too much to do; powerful enough tools would let faculty write most or all of their own courseware. This brings the CDG to the investigation of the possibility to construct more general authoring tools.

As mentioned before the courseware development heavily depends of the CDG. Nowadays some new ideas have come up related to the problems mentioned: Faculty members could be working with BASIC and (powerful) AUTHORING TOOLS coming down the way more and more. This would give faculty members control over the process again. In spring 87, an authoring system was in the Beta-Test at Dartmouth College. This AS could be used for all courseware projects, except very demanding concepts which should be implemented in a HLL. Using an AS the consultancy task will become more and more important.

To conclude: One aspect of the microcomputing project activities at Dartmouth College is the development of instructional materials - discipline specific courseware and generic authoring tools.

The development process emerged in two phases. The CDG finds this actual process to be a fruitful technique in collaboration between an instructor, usually a faculty member, and the CDG developer(s). This style makes good use of the college's limited people and monetary resources. The CDG made discoveries related to quality courseware development and is actually confronted with the situation that the faculty usually want to continue to make enhancements to a project long after support for that project has been exhausted.

ETC/UCI: A team approach for large scale CBL material development

The Educational Technology Center (ETC) at the University of California at Irvine (UCI) reflects eighteen years of continuous research and development in computer based learning. It is a little team of full time professionals in CBL material development which picks up additional staff for dedicated projects. The Center is not a service function of the University (no funds from UCI) and focuses on the development of sizable components of curriculum material.⁸²

The ETC has engaged in a spectrum of research projects connected with improving learning by the use of computers. Major (actual) interests include screen design, readability and comprehensibility of computer screens, user control of aspects of text display, individual versus group use of computers, cognitive consequences of computer use, methods of producing and evaluating CBL material.

The ETC describes the characteristic factors of its CBL materials as follows:

- Designed by competent teachers and researchers;
- Highly interactive;
- Individualized to each learner's needs;
- Use extensive graphics relevant to learning;
- Employ extensive spatial and temporal aids to readability;
- Use a variety of learner response formats;
- Emphasize natural language student response;
- Use a variety of pedagogical approaches;
- Contain computer simulations to aid intuition;
- Allow student control of the interactive dialog; and
- Estimable costs.⁸³

The materials are developed for a variety of environments (public libraries, museums, learning-at-a distance, elementary school through college/university, and commercial training) and cover physics, programming, teacher training about Educational Technology and a Scientific Reasoning Series. In addition, there are joint projects with the University of Geneva

⁸² Therefore the Center differentiates between trivial development, of individual courseware, and serious development of sizable components of curriculum material. The ETC is interested entirely in serious development.

⁸³ ETC/UCI, Statement of Capabilities, Irvine, March 1987 (Handout), p. 2

(Switzerland) [for instance, self-teaching spreadsheet]. Many older packages still are on timesharing systems (e.g. the mechanics course, electricity, relativity and astronomy packages).⁶⁴

Most projects emerge from internal proposals related to needs as Prof. Bork sees it. All projects are depending on funding. The latest project⁶⁵ is a joint effort with groups in the U.S. and Japan to begin the development of a course in Japanese, for English speakers. The course uses both computers and video-discs, to promote highly interactive learning environments for a wide audience. The support currently is from Fujitsu Ltd. and Nippon Television Cultural Society. Linguists and teachers of Japanese from many universities are cooperating in the project.

The ETC has the copyright for materials (not funded by a commercial institution). IBM has the exclusive right to run the Scientific Reasoning Series on its machines and market it.

Bork postulates a few basic assumptions of producing (large-scale) CBL material:

- The development of good learning material of any type is a nontrivial process.
- Learning material must be carefully evaluated and improved, in one or more formative evaluation.
- The development of good curriculum material is costly.
- In producing curriculum material a variety of talents are needed.
- Effective ways of producing computer-based learning material exist, and
- the ultimate test of any method of producing learning units including computer related material is the learning effectiveness of the materials produced.⁶⁶

In Bork's view, various production systems may all have the same fundamental components. These components may be arranged in different ways; some omitted. Furthermore, the decision of who carries out each activity and how the developers of the components communicate with each other will differ from production system to system;

- Preplanning activities;
- Production management phase;
- Objectives and specification phase;
- Training activities (for staff involved in production);
- Pedagogical design;
- Visual and graphic design;
- Technical development (implementation);
- Editing;
- Evaluation and Revision phase; and
- Marketing and Distribution (Dissemination).⁶⁷

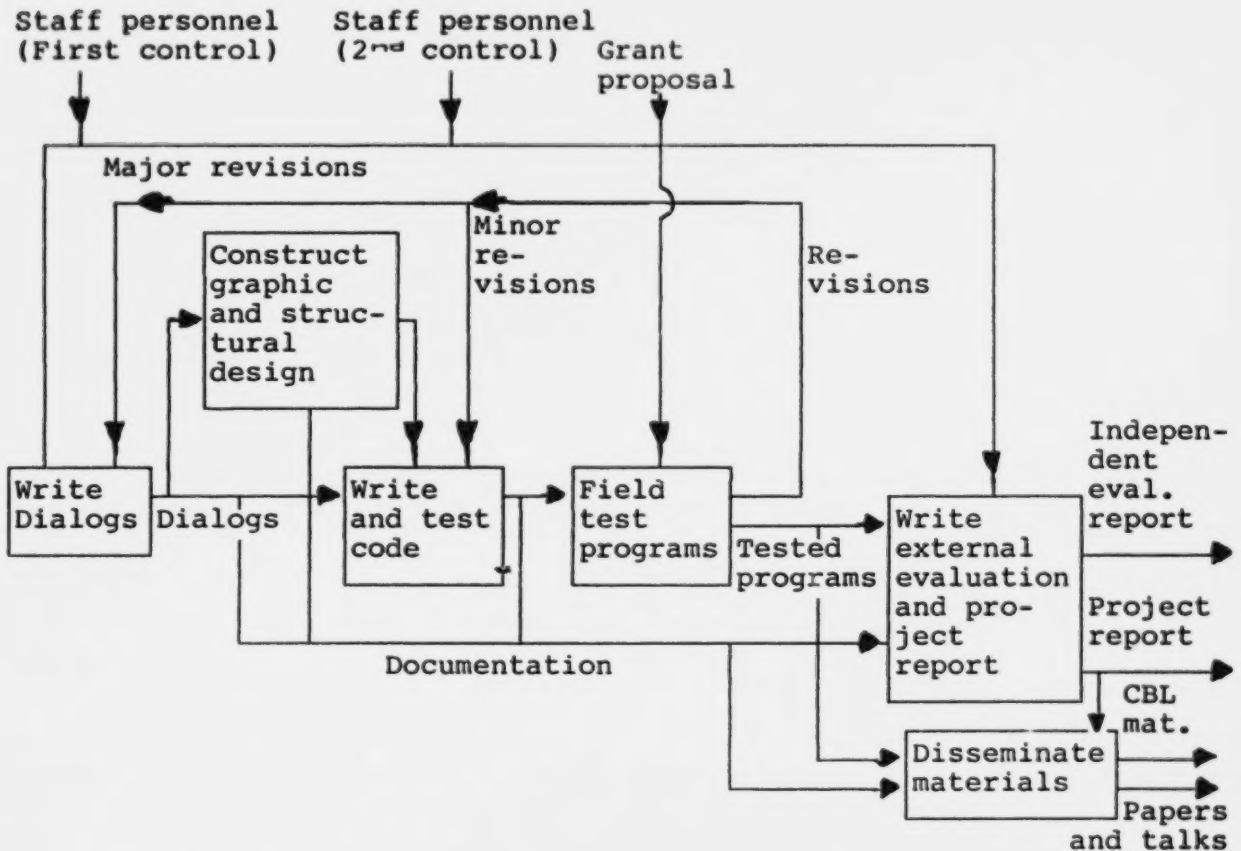
⁶⁴ These materials may be transferred to a PC environment but the ETC did not get funds for this change yet.

⁶⁵ Another project will be a pre-calculus course for university level, funded by the Annenberg foundations.

⁶⁶ A. Bork, 1984, pp. 178 - 180

⁶⁷ A. Bork, in: Walker & Hess (Eds.), 1984, pp. 98 - 100

The following illustration gives a top level overview of the Irvine production system focusing on the processes and products produced through the production process:⁸⁸



The Irvine production system can be described as follows relating to the above components and overview:

- Preplanning phase: At ETC this phase consists primarily of negotiating with the granting agency.
- Production management: Each project at ETC has a program manager associated with it. The procedures are based on SADT (see above overview illustration).
- Training activities: There are workshops and sessions for teachers, faculty members (to show them examples of different types of material), for graphic designers (to use the interactive design facilities), and for student programmers (with emphasis on the need for adequate documentation).
- Pedagogical design: The pedagogical design is broken down in two phases - the preliminary design and the detailed design phase. At the ETC this phase is regarded to be the most critical of all the stages in the production process.

⁸⁸ For details see A. Bork, in: Walker & Hess (Eds.), 1984, pp. 111 - 113

A group of teachers, a media specialist and a person who has done research in the particular area involved sit together to establish a set of descriptions of the computer modules, based on analysis of student needs (= initial pedagogical design). Brainstorming will generally be useful. The descriptions may be only a few pages in length, but they should contain enough details to guide the design group toward a reasonable view of the final product to design.

The group's structure for the detailed pedagogical design should be the same as for the initial design; the group(s) at the ETC spend one or a few weeks at this activity which is regarded to be very crucial for the quality of the final product.⁵⁷

To communicate between the group members during the pedagogical design and the programmers (for implementation), a recording method called a **SCRIPT** is used. This documentation method emerged at the ETC through time and experience and is found to be easy comprehensible. The script for a learning module must contain all details necessary: Messages on screen(s), testing of student input and its sequence, overall sequence and structure of the module, and comments (for programmers).⁶⁰

- Visual/Graphic design: Ideally visual and graphic design should be done by a competent graphic designer who does magazine layout or develops brochures advertising products. In practice, this design is often done by a combination of the pedagogical authors and programmers.
- Implementation: Before any actual programming takes place, structure charts are developed. In addition there exist standards for structured programming. At the ETC Pascal is used for implementation in conjunction with extensive auxiliary software, for instance an input-output package, and so on. In addition, a library of procedures for the commonly carried out functions is useful. While there is an idea of using a code generator who could generate up to 80 % of the code by its own (from the design documents), this generator needs to be developed by the ETC first.⁶¹
- Review/Evaluation and revisioning: Reviewing and evaluation of the product in progress has to take place at several stages. The first time after the pedagogical design, then after initial coding and finally after the field test. A product may be evaluated and revised several times during its development until it is ready for dissemination. For this stages standardized (question) checklists exist at the ETC.⁶²

⁵⁷ For details see: A. Bork, Pedagogical Development of CBL Material, in: A. Bork and H. Weinstock (Eds.), Designing CBL Materials, NATO ASI Series, Vol. F 23, Berlin/Heidelberg, 1986, pp. 67 - 94

⁶⁰ A. Bork, 1985, p. 129

⁶¹ The Center began to seek funds for this project.

⁶² For examples see: A. Bork, 1985, pp. 138 - 141

To conclude: The ETC at UCI employs a team approach which pays attention to the fact that the pedagogical design is a very critical step in CBL material development. For developing quality and sizable components of curriculum material, the Center wants to collaborate with good teachers (= instructors who are aware of the interactive character of instruction), give the design team freedom in its work (with the exception of the "script" as a standardized document on which the design is recorded because the result of the design must be usable for the programmers), and is aware of the editing, revision and evaluation function so that a program can be revised several times until it is in a stage for dissemination.

Howard University: Faculty driven courseware development

While other institutions are investigating the use of authoring tools (AT), authoring languages (AL) and Authoring Systems (AS), Howard University at Washington, D.C. already made its commitment to work with CMU Tutor as an implementation environment for faculty and did get started producing its first (small) prototypes with CMU Tutor after having made the first experiences implementing a few educational software packages with an all purpose (high level) language.

How did the overall process and involvement of faculty start? The idea is to have a few initialization projects to demonstrate the results to faculty members of the entire campus. The overall claim is to motivate faculty to start with CMU Tutor, too.

In 1984 (when the ICEC concept was announced) 10 faculty members were actually submitted for a concept proposal out of 84. Three out of these have been chosen for a first project round. Since CMU Tutor was not available at that time, these faculty members had to have substantial programming experience and developed their materials with an all purpose (high level) language (HLL). In the spring of 1986 the University was asked for nominations for ICEC⁶³ Fellows. Of the 12 who were nominated to ICEC, three were selected as Fellows and four as alternates. Finally five had the opportunity to attend an introductory course to learn CMU Tutor at Carnegie Mellon in summer 1986. In October 86 Howard University received the first machines (IBM RT) to start the initialization projects. Examples of these projects focus on little children's perception of letters, a package for descriptive statistics and a chemistry course. One of the projects is supposed to be finished by fall of 1987.

A Courseware Development Lab was established by the University's ICEC Committee to provide a facility in which faculty members can develop courseware packages. Actually the lab is getting some more machines.

A forum was established which takes place on a weekly schedule because the development of courseware is decentralized and in-

⁶³ InterUniversity Consortium for Educational Computing (initiated by the Carnegie Mellon University in 1983). For details see K. Adler's report.

dividualized - each faculty member implements its own design concepts and ideas. At these meetings the faculty member(s) share ideas, discuss pitfalls and progress of projects (coordination), and help each other. This forum seems to be very important for the faculty and the coordinator (from the Office of the Vice President for Academic Affairs).

The "novices" of the faculty are pleased to work with CMU Tutor instead of having had to learn a HLL. In addition they feel in control of the process through the AL; they can make it happen their way. On the other side faculty member(s) who already did develop programs in a HLL (for instance, in Pascal) had to relearn another environment. It is a new orientation for these faculty members.

To conclude: Howard University is one of the institutions which made a commitment to start courseware development with an authoring language environment. Last summer a first group of faculty members had the opportunity to attend an introductory course of CMU Tutor in which a few projects are now developed to motivate faculty of the entire campus. As the projects are in their initial stages, a final judgement on the procedures to develop the materials and the courseware itself cannot be made, but it will be interesting to follow the activities at Howard University.

The informal small team approach at the University of Texas (Austin)

There is a long tradition of courseware development and use at the University of Texas at Austin (UTA).⁶⁴ This tradition emerged from instructional design and was adapted in the late sixties for computers. In the seventies, the University of Texas (at Austin) became a member of CONDUIT. Due to this long time involvement and experience evolved a systematic approach to courseware design related to the instructional design of materials. The spreading of personal computers in the eighties resulted in a new effort established in 1984: PROJECT QUEST.

PROJECT QUEST is a long-term effort of the university to increase the innovative use of microcomputers in teaching and research. It is supported in part by gifts of microcomputer hardware and software from donor companies. The project resulted from a planning committee report in 1984. A special review committee (10 - 12 persons) was established for the project. Besides this committee, there can be found special QUEST staff [assistant director of the Computation Center for Instructional Computing (as a coordination function), educational special project assistant(s), hardware maintenance assistant(s), and so forth.]. Finally the Computation Center has also established a special consulting group on microcomputers to serve QUEST projects and other microcomputer users on campus.⁶⁵

⁶⁴ 46000 students are enrolled at the University of Texas at Austin, including the graduate students.

⁶⁵ For details see: Progress Report, PROJECT QUEST, University of Texas, Austin, December 1986

In May 1984 the PROJECT QUEST committee issued its first call for proposals to the faculty. Since then, the committee has received more than 600 proposals. Of that number, 389 had been approved for equipment funding until end of 1986. Most projects awarded so far have been small; the average number of machines per grant is 3.3.

Colleges submitting the most proposals and receiving the most equipment are Natural Sciences, Engineering, Liberal Arts and Business Administration. They have been awarded 74,9 percent of all awarded so far. Of the machines awarded, 70,4 % are being used for teaching; 20,3 % is used in research application and the rest (9,3 %) is employed in joint teaching-research projects.^{ee}

The following list serves as an illustration of the wide variety of projects within PROJECT QUEST:

- CAI simulations of experiments in chemistry students also perform in the beginning chemistry lab. (Teaching project no. I003)
- The Center for Teaching Effectiveness PC demonstrates instructional systems design and CAI modules in a wide variety of disciplines. (Teaching project no. I002)
- Astronomers develop software to analyze high resolution stellar spectra and link to mainframe computers to compare real-time observations with theoretical synthetic spectra. (Research project no. I034)
- Transport engineering students employ PCs to simulate traffic flow, optimize signal timing, plan roads and design pavements. (Joint project no. I052)
- Beginning biology students use the PC to take quizzes, receive remedial tutoring, perform game-like simulation exercises, and learn how to analyze data. (Teaching project no. I063)
- Students of russian literature will benefit from a set of CAI materials designed to aid reading and translation. (Teaching project no. I159)
- Maps of plant populations are displayed on the screen and manipulated directly in this study of how the spatial structure of plant population changes over time. The model permits graduate students in biology to conduct "experiments" in plant population dynamics. (Research project no. I188)
- A small lab aids bilingual education students in improving their written language skills in Spanish. CAI instructional software for written Spanish will be tested and modified. (Teaching project no. A130)

More than half of the instructional software packages which were or are developed can be regarded as simulation-based educational games, tutorials, drills, and tests.

The general production process of (instructional) software is split up as follows:

- All ideas coming from faculty are subject to a review: Everybody has to establish a (written) proposal in which the

^{ee} For details see: Progress Report, PROJECT QUEST, University of Texas, Austin, January 1987

concept of the project is outlined. The review committee decides which proposals will be supported by QUEST with hardware, software, and support.

- The faculty member(s) meet(s) with QUEST staff to discuss the development of the software. Heavy attention is given to the instructional design and a systematic approach at this stage of the project.
- The faculty member(s) in concern designs the material (especially the sequencing of the material) - mostly having in mind the systematic design approach. There is no formal format for the design phase. As a matter of fact, from flowcharts to storyboarding, frame structuring to direct flow indications can be seen everything. The technique to record the results of the design is not as important as the process itself; the point is to follow the steps of a systematic approach!

The overall experience within PROJECT QUEST showed - as a QUEST coordinator sees it - that most faculty had no training in instructional design and/or following a systematic approach but they knew (intuitively) how to sequence the process and the presentation of the instructional material.

- The designer (= faculty member) gives the result of the design stage to a designated programmer (mostly a graduate programmer) which codes the features of the design. There is not a recommendation to use a certain HLL. The faculty member(s) and their programmers are free in the use of any language. More than 50 % of all software is written in BASIC! This language comes automatically with the IBM PC and is the most common language known by people. Some projects are coded in PILOT (an authoring language).
- After (a) final review and revision, the software is used by the faculty member(s).
- There are no summative evaluations of instructional software by QUEST staff anymore. At the beginning of PROJECT QUEST such evaluations were conducted but in the meantime this process was stopped because it was found that (nearly all) the materials can really have additional functions worthwhile for teaching. But it is up to the faculty member itself to conduct such summative evaluations for his concern.

The proposal review stage is a very hard and serious one in conjunction with (a) faculty member meeting(s) to make clear the importance of a systematic approach. If the faculty member is not aware of this important aspect, there is danger that it will be a weak project.

There is no formal evaluation of the projects, but after 12 - 18 months a QUEST staff member evaluates what was done with the (donated) hardware and software through an "Informal Project Review" (Self evaluation of projects). Questions within this project review focus on the hardware and software environment, the personnel assigned to the project and the (adequate) progress of the project itself. A recent evaluation (end of 1986)

shows that the average satisfaction level for hardware received is 88 %, and for software 78 %. The average QUEST project was 47 % complete as of January 87 (134 projects totally evaluated). While continuing to support seed projects, the Committee [= review committee] increasingly devotes its efforts to seeing that new teaching materials for microcomputers have an opportunity to succeed or fail in larger laboratory and classroom settings.⁶⁷

It is interesting that most of the (instructional) software developed is only for local use and most faculty have no commercial focus. The copyright is still a "grey area" - a concept is now in press. PROJECT QUEST requests all (instructional) software developed on campus (with hardware donated through PROJECT QUEST) to be free accessible for all other departments on campus.

A coordinator of PROJECT QUEST characterizes this production process and the production strategy as informal (small) team approach. The University of Texas at Austin has come to this approach because the team strategy has very high costs. The informal team is a good approximation of a team strategy without having high costs. To make clear the difference between the "traditional" team and the informal team approach: At Austin there is made a clear point on the systematic approach to develop courseware at the proposal and discussion stage with the faculty members. But besides, faculty are free to develop the courseware on their own (with the designated programmer). If they need any assistance, specialized QUEST staff is available. In a "traditional" team approach a variety of specialists are designated to one project and work just on this project - stage by stage - until it is finished. Mostly this approach is more or less formalized and characterized through documents as means of communications in all stages of the production process.

To conclude: PROJECT QUEST at the University of Texas at Austin can be seen as a project which is strongly driven by faculty and their ideas. The QUEST staff, support groups and review committee are here to guarantee the general direction of the project and to coordinate all the widespread activities going on at the campus. It seems that this "informal team approach" where the developers are free in the implementation is the result of the number and growth of projects within PROJECT QUEST. In my opinion, this developer's freedom is possible as some of the faculty members already made their experience with main-frame courseware development in the sixties and seventies on one hand and through the serious review of the proposal where the "development success" of a certain project is subject to discussion on the other hand in conjunction to faculty's confrontation with the systematic approach to ensure a certain quality of the instructional software to develop.

⁶⁷ For details see: Progress Report, PROJECT QUEST, January 1987

Stanford University: Removing obstacles for (courseware) development

Due to (mainframe) computer shortage for students in 1983 and the appearance of microcomputers on the market, IRIS (Instruction and Research Information Systems) was created and designed to support the Stanford University community in the transition towards a microcomputer-supported environment. IRIS is a division of the Vice Provost's Office for Academic Computing and Information Systems (ACIS)⁶⁸ and disseminates information through various channels. IRIS is trying to integrate the Computer Science Department and the Main Computer Services into the project.⁶⁹ IRIS provides computing advice and information to faculty, students, and academic staff. In addition to maintaining a variety of support programs and activities, IRIS initiates and manages special projects designed to determine and to promote appropriate uses for computing at Stanford University.

Besides the traditional computer science activities, a bunch of activities in instructional computing can be found at Stanford University. This has become evident as an increasing number of faculty were using computers as an instructional medium in their courses. Special on-campus projects and Stanford's membership in several university consortia have provided the impetus for integrating computing and instruction.⁷⁰

The Faculty Author Development (FAD) Project and Project Ramus⁷¹ are special projects within Instructional Computing. IRIS was responsible for the Faculty Author Development (FAD) Project and is now initiating the Courseware Authoring Tools (CAT) Project, too. Begun by the Office of Academic Computing and Information Systems (ACIS), the FAD Project was chartered with encouraging faculty interest and experience in developing instructional materials on a computer. The 2 1/2 year FAD project was funded primarily by donor companies and had the goal to create a showcase of models from the faculty's courseware projects. FAD began in 1984 with the appointment of a manager and the hiring of professional and student programmers. Since a major goal of FAD was to seed interest in and develop knowledge of microcomputer courseware among faculty, IRIS chose proposals (see next section) from a wide range of schools and departments. As a result, since the fall of 1984, faculty members have been using microcomputers as an instructional medium in courses spanning 12 disciplines. In all, 38 professors from 26 departments participated in the FAD project.⁷²

The production strategy can be compared with "academia's basic module": A faculty member works closely with a programmer.

⁶⁸ ACIS oversees and plans for all aspects of academic computing at Stanford. One of ACIS three divisions is IRIS.

⁶⁹ For details see: K. Adler, 1986, pp. 83 - 91

⁷⁰ For details see: About Computing at Stanford, A Guide for Faculty and Students, IRIS at Stanford University, September 1986

⁷¹ For details on RAMUS see: K. Adler, 1986, pp. 87 - 88 and About Computing at Stanford, 1986, pp. 9 - 10

⁷² For a detailed description of projects see: Creating Courseware, FAD Project, Stanford University, October 1986

Faculty represents the subject matter expert. Hidden parts in faculty are the pedagogical expert and the visual/graphics design expert. While for many packages this strategy worked, there were packages as the "Would be Gentleman" and "The Theater Game" for which a graphics designer was hired!

The production process for creating courseware within FAD can be described as follows:

- By spring 84, and again in winter of 85, letters were sent to all faculty inviting them to submit proposals to participate in the FAD project. The faculty members were asked to outline how they would use microcomputers in their courses, assuming they were provided with equipment and programming help. A steering committee accepted proposals which included a novel and useful application of microcomputers in the professor's course. A key consideration was whether the proposal could be turned into a working program in less than a year, given the available resources.
- Two levels of support were offered for the proposals accepted as FAD projects. For those projects in which the professors had the knowledge and resources to develop courseware on their own, IRIS provided the hardware and some programming support. For other projects, IRIS provided both hardware and the programming support. (Professional and/or student) programmers were working closely with faculty members to develop courseware.
- The design stage was very informal. Normally there have been meetings between programmer(s) and the faculty member(s). As a result of this (brainstorming) meetings the programmer outlined a specification which was reviewed with the faculty member.
- The next step was the implementation of some prototype(s), a small simulation, some screens were created and coded to show faculty the appearance on the microcomputer.
- After reviewing this prototype, the coding of the entire course was done. Again the communication between the programmer and the faculty member was very crucial during this stage.
- After a final (internal) review and/or revision, the courseware is used by faculty members in their courses. In addition most of the packages are disseminated through the Office of Technology Licensing (at Stanford) and/or through Kinko's Academic Courseware Exchange (ACE).

When the FAD project came to an end in October 1986, plans had already been made to carry on the work begun by FAD. A new project, Courseware Authoring Tools (CAT) is now underway. CAT's goal is to provide Stanford faculty members with a set of powerful, easy-to-use authoring tools that will minimize the coding effort needed to develop courseware. These tools should be as easy to use as MacWrite or Visicalc (appearance of the programming environment) and as immediate as the Pinball Construction Set. This will give faculty more control of the courseware development process and allow them to be the primary

author of courseware. It is an approach to remove obstacles so that faculty members can control their use of computers. The environment designed through CAT should be flexible enough to allow real freedom in courseware design, with 90 percent of the work done by the faculty member and only 10 % done by a programmer. As with FAD, Stanford professors will be involved from the onset of the project, providing input on the design of these authoring tools. One of this tools actually developed for advanced machines is a simulation tool for sociology; it is hoped that this first tool will be finished by summer 87.

To conclude: The overall idea of applying (micro) computer technology is to enhance academic achievement. There is a wide variety of activities going on at Stanford University. One of the special projects was the Faculty Author Development (FAD) Project to create models and prototypes to show academia the potential use of courseware in courses. FAD's goal was the cultural change, the spread out of showcases. In the meantime the Courseware Authoring Tool (CAT) Project has begun to involve even more faculty members and to give them tools for (advanced) machines. This should give faculty more control of the courseware development process.

The development of a support model at the University of North Carolina at Chapel Hill (UNC-CH)

The UNC/IBM Courseware Development Project (CDP) at the University of North Carolina (UNC) at Chapel Hill was jointly funded in 1983 by the UNC College of Arts & Sciences and a computer equipment company (IBM). At present, there are approximately 40 individual instructional software projects,⁷³ in many different disciplines, and in all stages of development.

Looking at the history and development of the CDP, two phases can be distinguished: A starting phase (= Phase 1) beginning in 1982/83 which was characterized by negotiations and starting the first projects with faculty members. At that time the faculty member and the (student) programmer were the primary developers with general microcomputing support from one full time staff member. Very little instructional design support was available. This meant that the design, evaluation and the project management were primarily the faculty member's responsibility! Some faculty members used PC Pilot to code the instructional software. Others used BASIC and PASCAL.

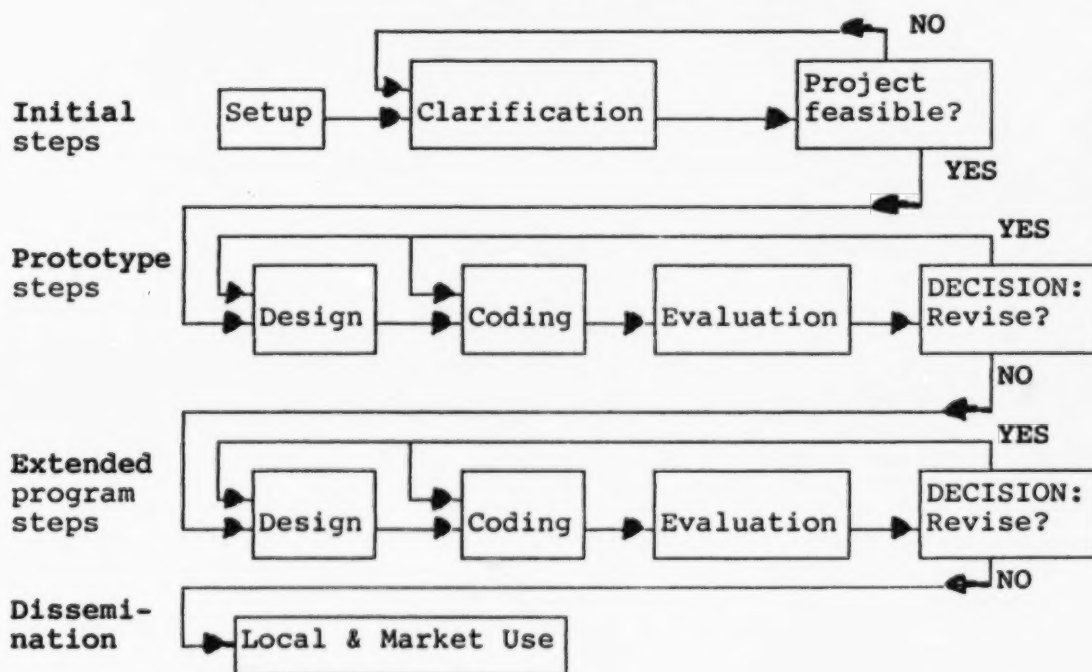
In 1985 the CDP became more institutionalized and established (= Phase 2) as a result of experiences developed during the first phase of the project. Instructional software production has changed considerably since its beginning in phase 1. Phase 2 is characterized by improved planning combined with increased staff and resources. The main programming language is now TURBO PASCAL.

The production strategy approach for each project of phase 2 is based on a faculty developer (the team leader) and a (student)

⁷³ For a description of the courseware projects see: Descriptions of Courseware Projects, CDP, UNC, Chapel Hill, February 1987

programmer. The basic two-member development teams are supported to varying degrees by professional staff consisting of a project coordinator, an analyst programmer, and an instructional designer. The extent of the professional support staff's involvement in any particular project is affected by two factors: First by the faculty member's time and priorities, and second by the fact that the support staff work with many projects simultaneously. All student programmers are closely supervised by the CDP analyst programmer; other support staff members operate as occasional consultants to individual projects. It is the CDP staff's opinion that the autonomy and part-time nature of individual projects make the establishment of standards and procedures relatively difficult (in academia). Fortunately, an occasional but intense software review, initiated and performed by the professional staff, has proved to be suitable to faculty members, when continuous involvement of the staff cannot be.

The development process (of phase 2) is diagrammed below:⁷⁴



- A faculty member who is interested in the development of courseware may receive hardware and (generic) software, and will be trained in the PC's operating system and a word-processor. This is done through the setup phase. After a faculty member's project is approved, he/she will have several meetings with CDP staff to clarify the project idea. The information gathered during this clarification phase will help to make decisions about the feasibility of a project and the al-

⁷⁴ M. Anderson, Instructional Software Development: Procedures And Resources, Chapel Hill, June 1986

location of resources.⁷⁵ The clarification phase will take from a month and a half to a semester.

- After the establishment of an accepted project concept, the faculty member spends several months deciding precisely how to implement his/her ideas. Decisions that will take place with the help of an instructional designer and/or other CDP staff include the type of instructional software (tutorial, simulation, tool, etc.), the overall courseware structure, the layout of individual screens and the programming language. It is requested by the CDP staff that all the design ideas are recorded in some form. The instructional designer's preferred tool for screen layout is actually Dan Bricklin's DEMO program. At the end of this first design phase the following products have been created: A clear definition of the objectives and the target audience, a comprehensive outline of content material, a diagram of the structure of the courseware, screen-by-screen plans for a prototype including an outline of supplementary documents (such as an instructor/student's guide and a description of proposed evaluation methods). The production of first design materials may take from six weeks to six months, depending on the complexity of the project, and the amount of time and prior experience of the faculty member. To ensure consistent programming across projects and good program structure, an experienced programmer is assigned to the project for a semester to develop the overall data and file structure and the logical flow. Next, a student programmer trained by CDP staff will code the prototype.⁷⁶ Formative evaluations are done throughout project development to identify areas which should be revised to make them usable and instructionally effective. This is an iterative process and it is recommended by CDP to do initial evaluations early, when the prototype is still in a rough form.
- The same cycle is gone through again for the extended program: Extended design, coding and evaluation.⁷⁷ Most developers have benefitted from conducting student evaluations over a period of two to three semesters so that revisions made after one semester of testing can be tried out with students the following semester.
- After the courseware package is completed, it is ready for classroom use. To meet a publisher's standards, it may well be necessary to revise the material again. CDP staff attempts to accommodate faculty members' needs for revision and program maintenance.

⁷⁵ Decisions at this stage include whether the microcomputer is an appropriate medium for the proposed instructional project, how much and what kind of CDP staff support will be needed, and how the final courseware will be used.

⁷⁶ These student programmers are supervised through bi-weekly meetings with CDP staff.

⁷⁷ For a detailed description see: M. Anderson, 1986, pp. 6 - 7

The CDP staff identified a list of impediments to rapid progress:⁷⁰

- Inadequate design;
- Excessive programmer turn-over (more than 3 programmers during the life-span of a project);
- Lack of programmer supervision and/or lack of communication between programmer and CDP staff;
- Code produced by team members other than the assigned programmer;
- Inadequate communication between the faculty member and CDP staff; and
- Lack of periodic evaluation by students and other reviewers.

The prototype phase is seen as the very heart of their production system and as a method to ensure good quality courseware production. If a prototype phase would not exist, revisions must be made after the program is complete and an evaluation has been performed. This alternative is more time-consuming for all team members. Prototyping was developed in three phases during the lifetime of CDP: The first phase was simply writing down notes, sketches, or an outline on paper. The second phase was designing individual screens with the help of a word-processor. The third phase includes the first two, but is supplemented by design tools such as Dan Bricklin's DEMO program⁷¹ and PC Storyboard. It is the CDP staff's opinion that these tools help a great deal in the design of the courseware.

To conclude: The UNC-CH CDP works with a faculty-centered team to develop courseware (with a faculty member as team leader and a student programmer). But to ensure certain standards over all the projects, it is required to have tangible incentives and clear signals from a centralized group - the Courseware Development Project (CDP) staff, which consists of professional personnel for instructional design, programming and project management. The underlying procedures for courseware development evolved over time and represent an approach and strategy which can be called a "support model". It is the CDP staff's opinion that with the present support model, better courseware is being produced with less faculty and programmer time and less wasted effort, compared to the approach underlying phase 1.

3.2.2. AUTHORING TOOLS/LANGUAGES/SYSTEMS AND THEIR ROLE IN COURSEWARE DEVELOPMENT

Trade off discussion

A very popular discussion focuses on problem and benefits of using authoring tools, languages (AL), and systems (AS). Bork, for instance, for years already argues that the two problems with single-individual systems are closely tied together in an unfortunate fashion: If the authoring language (AL) is extremely simple, then it can be, as advertised, quickly learned by some teachers. But it can only write (very) simple-minded (and small) units. As the AL becomes more complex, increasing the

⁷⁰ M. Anderson, 1986, pp. 7 - 8

⁷¹ DEMO supports the design by giving the opportunity to design screens of a prototype without generating code ("vaporware generator"). It helps to outline the faculty member's ideas but it does not handle graphics and is just an editor.

range of [large-scale] CBL material possible, it becomes less and less usable by naive individuals. Another difficulty with production strategies of this type comes in connection with revision activities (Trade off between flexibility and power).⁸⁰

Marsh developed a small series of courseware with Super-PILOT⁸¹ for middle school students to investigate the feasibility of instructor use of an AL. The general finding was that the use of the authoring language was so time consuming that it does not appear a feasible method for educator-developed courseware. In Marsh's view the regular classroom teacher or the local school administrator will not be likely to take part in the development (implementation) of courseware. In addition she argues that learning and writing a lesson takes a great deal of time and even more time to debug and correct a lesson. Finally she sees the danger of the electronic page turner.⁸²

Tarp addressed a similar question: Are the instructional properties of CAI modules produced using the IMAS template type authoring system⁸³ different from the instructional properties of CAI modules produced through a traditional introductory course in instructional design and basic programming? One of his findings is that the template AS used requires a reduction in planning flexibility, but offers a significant advantage in terms of authoring efficiency (less time for AS than with programming language, author's perception of ease of use) and instructional quality (design attributes, cognitive level).⁸⁴

Who wants to develop what under which circumstances?

While Bork seems to have in mind the production and maintenance of large-scale courseware, and Marsh the effectiveness of an AL against conducting the content through traditional instructional strategies, Tarp compared the template AS against the development of courseware with a programming language (Basic). This highlights an important aspect of this discussion: WHO should work with AL and AS for producing WHAT (kind of courseware) under WHICH constraints and circumstances?

⁸⁰ Alfred Bork, in: Walker & Hess (Eds.), 1984, p. 103

⁸¹ This is an AL for the Apple Computer.

⁸² M. Marsh, CAL in the Social Studies: Development and Evaluation of a Series of Computer Programs for Middle School Students, Diss. (University of Maryland), 1984 [UMI No. 85-12193]

⁸³ The University of Texas Instructional Module Authoring System (IMAS) is designed to operate on Apple II microcomputers, and assists nursing education authors. The template for the AS provides authors with a pre-specified instructional sequence for diagnosis, planning, and evaluation (= components of nursing process).

IMAS evolved as NEMAS = Nursing Education Module Authoring System. For details see: S. J. Grobe, Nursing Process Algorithm Used for a Nursing CAI Authoring System, in: Medinfo-83 Proceedings, J. Van Bommel and M. Ball (Eds.), Amsterdam, 1983, pp. 194 - 197

⁸⁴ J. Tarp, A Comparison of Instructional Design Characteristics of CAI Lessons Produced by a Computer-based AS and by Traditional Design and Basic Programming, Diss. (University of Texas), Austin, 1982 [UMI No. 82-27729]

Heines points to the same question by saying that AS (only) address the programming aspect of producing instructional software, not the design. Even the latest generation of symbolic AS⁶⁵ is in this line. The design step becomes more and more important because through a "faster" language/ system everything can be implemented so quick!⁶⁶ In other words: AL and AS try to reduce the programming complexity, but still require the author to provide the instructional design.

Criteria for ideal AS and choosing AS

This guides to the question what properties should have an ideal authoring system that guides the instructional design? Merrill states the following prescriptions for such an AS:⁶⁷

- 1) An authoring system should provide the author with a wide variety of instructional transaction archetypes⁶⁸ (for presentation and practice, test) which serve as models for student/program interactions.
- 2) An authoring system should provide the author with a selection of strategies for combining transactions.
- 3) An authoring system should provide the author with a selection of content structures for organizing the subject matter.
- 4) An authoring system should provide the author with tools to specify student management parameters.
- 5) An authoring system should provide the author with various levels of interaction including templates, parameter selection menus, and template creation.
- 6) An authoring system should provide guidance to the author in the selection of goals, content structures, strategies and learner management parameters.⁶⁹

Hazen classifies her criteria for choosing an authoring tool after the staff functions in instructional software development, that are criteria important to the content specialist, the instructional designer, the programmer, and the instructional software administrator.

- a) An authoring tool for a content specialist should
 - provide prompter or menu-driven software to aid the writing and revising of programs if a programmer is not available, and
 - provide excellent documentation (tutorials, on line help, sample programs, reference/technical manuals, reference cards).

⁶⁵ For instance, Action in course (for the Macintosh) or Maestro (for the IBM PCs).

⁶⁶ J. M. Heines, Authoring Productivity Revised, in: Training News, Vol. 8 (1986/87), No. 7 (March), pp. 8 - 9

⁶⁷ The aspect of the production strategy is not addressed in the article.

⁶⁸ Merrill does not work with the FRAME assumption, but uses the transaction concept.

⁶⁹ M. D. Merrill, Prescriptions for an Authoring System, in: JCBI, Vol. 14 (1987), No. 1 (Winter), pp. 1 - 10

- b) An authoring tool for an instructional designer should
 - provide features to encourage highly interactive software, and rapid response times;
 - facilitate alternative modes such as drill and practice, simulation, as well as tutorial software;
 - provide graphics capabilities and graphics editors;
 - provide sound or speech synthesis capabilities and appropriate editors
 - provide alternate input modes;
 - provide structural features such as help and hint routines, skip and restart routines, glossaries and overviews;
 - provide windows or split-screen effects, and
 - provide a variety of screen output formats.
- c) An authoring tool for a programmer should
 - provide sophisticated answer analysis or judging for constructed or open ended responses;
 - provide rapid "revise, test, and revise" cycles;
 - provide powerful features, such as graphics capabilities, interactive modes, randomizing functions, subroutines, screen control for erase and write of whole or partial screens, and so forth;
 - provide a tool already familiar to the programmer, and
 - facilitate maintenance programming
- d) An authoring tool for an instructional software administrator should
 - provide management or administrative instructions for collecting student/system interactions and data;
 - offer technical help (hot line, reference manual);
 - be cost-efficient;
 - can be uploaded to a mainframe, or to a networked hard disk, for centralized updating and distribution purposes; and
 - provide the ability to run the instructional software packages on several hardware devices.⁷⁰

These criteria can be a starting point for the selection and use of an authoring tool appropriate for the project(s) at hand. It seems to be clear that it is extremely difficult to find all of the criteria in one tool. Furthermore the computer is used in such a large variety of ways that no one authoring tool can be expected to meet all the standards for each project's needs.

Situation on the marketplace

Besides the discussion of problems and benefits of authoring tools, and the establishment of criteria for the choice, more and more (new) AL and AS can be found on the marketplace. At the 5th CBT Conference⁷¹ for instance, this fact is reflected as follows: In 1982 (at the 1st conference) nine vendors offered twelve AS, last year there were 68 AS, this year 93 AS presented by 150 vendors. In other words: The field of AL/AS has grown every year, and shows no signs of abating. But it is

⁷⁰ For detailed description see: M. Hazen, Criteria for Choosing Among Instructional Software Authoring Tools, University of North Carolina at Chapel Hill, Chapel Hill

⁷¹ This conference has taken place in Atlanta in March 15 - 18, 1987

not only the number of AL/AS that keep growing, either, it is also their capabilities. The latest news are the shell²² and symbolic authoring systems. Most of the systems are micro or mini-based, but there exist (still) a lot of mainframe-based AL/AS, too.

Symbolic authoring consists of two steps [i) building the structure of a course and ii) filling in the content] and uses icons to represent the structure and pop up windows to fill in the content. This kind of AS works with one rule: select an object and perform an action on it.

Conclusion

The selection and the use of authoring tools as AL and AS heavily depends on the staff who has to work with these tools (WHO aspect), what has to be developed (WHAT aspect) under WHICH circumstances and constraints.

For the time being, the marketplace offers more and more AL and AS, and the capabilities are growing. This situation requires that the environment, the instructional software to be developed and the staff available is to consider in choosing an authoring tool for projects at hand.

At first glance, it seems that for

- small-scale in-house (local) individual courseware production, and having a
- need of courseware that is highly tailored to local interests, and to
- create specific modes of instruction (i.e. drill, tutorial) which cannot be found on the market,

AL and AS can be very competitive and have benefits.

For large-scale team-based production of courseware with complex features, the all purpose (high level) language (HLL) with adequate tools (still) seems to have more benefits.

It is our opinion that the future development will either shift the usability and capability zone of AL and AS more and more towards the all purpose languages, and/or the all purpose language environments may be supported with more and more authoring tools to shift towards AL and AS and therefore the question of the trade off between all purpose language(s) and AL/AS may become more and more irrelevant.

3.2.3. SUMMARY AND CONCLUSIONS

The purpose of this chapter was the presentation of a variety of production approaches and strategies. As a matter of fact, main properties and features of twelve production approaches and strategies (in alphabetic sequence) were described. We visited nearly all of the sites outlined during an informative trip through the U.S with the exception of the collaboration model which was outlined based on an article. Additional places

²² System in which one program acts as an environment for another program. A shell program may "wrap" around another program, providing a user interface or customized environment for the user.

For detail see: W. Judd, Creating Lessons with Shell AS (Session 605), 5th CBT Conference, Atlanta, March 18 - 23, 1987

were visited, but are not described in this chapter as they represent an approach and strategy similar to another already outlined and/or primarily focus on other aspects than the production approach and strategy itself.

Additionally a short overview was given on authoring languages (AL) and systems (AS). An important aspect for the evaluation of such tools is the fact that these tools (try to) reduce the programming complexity, but still require the author to provide the instructional design. AL/AS can be an appropriate production environment depending on who has to develop which software under which circumstances and constraints. It may be that in future times the trade off between AL/AS and all purpose language(s) may diminish due to the beginning trend that the AL/AS environment(s) as well as the all purpose language environment(s) are shifting towards each other.

3.3. IDENTIFICATION OF DEVELOPMENT CHARACTERISTICS AND CLASSIFICATION OF DEVELOPMENT APPROACHES AND STRATEGIES

In the preceding subchapter a description of various and different production approaches and strategies was given and the role of authoring languages and systems outlined. This subchapter focuses on the establishment of development characteristics as a helpful instrument to identify (quality) enhancement factors and to assess and classify production approaches and strategies.

3.3.1. IDENTIFICATION OF DEVELOPMENT CHARACTERISTICS

Introduction

The overall purpose of this report mentioned in the first chapter is the identification of (quality) enhancement factors for the development of courseware which (can) discriminate between good and bad courseware. The starting point behind this approach is the assumption that the production approach and strategy influences the quality of courseware itself.

We selected development characteristics as an instrument by which development approaches and strategies can be described and classified. But our primary concern is not the classification of production approaches and strategies, rather the identification of (quality) enhancement factors for the development of courseware.

By development characteristics we mean forming parts and special qualities belonging to production systems which make it possible to describe and classify production approaches and strategies. Sometimes these development characteristics are called "procedural guidelines", "procedural informations", or "properties", too.

Literature review

As a matter of fact, not much literature on development characteristics could be identified, though this may be a good instrument for describing and classifying production systems. As the term "development characteristics" could not be found in

literature, we identified some synonyms which seem to describe the same content. Major contributions were found in publications from Bork, Mitchell, and Smith.

Bork⁷³ describes the following "properties" of production systems:

- Major focus on pedagogical issues;
- The production system should produce the best possible material;
- Reasonable costs;
- Estimatable costs;
- Involve a wide range of good teachers;
- Easily revisable units;
- Encourage interaction;
- Encourage individualization (of courseware);
- Maximize motivation (of courseware);
- Minimize transferability problems, and
- Adaptability to large-scale production.

Mitchell⁷⁴ states some "procedural comments" for the development of courseware:

- Working from a plan for a set of programs (rather than designing each independently);
- Beginning of the programming (after considerable planning of the pedagogical design);
- Determination of program design (by the educational objectives);
- Integration of software quality standards (followed rigorously);
- Establishment of an unending maintenance loop (should be a major part of courseware development); and
- Evaluation and modification of courseware (repeatedly).

Smith⁷⁵ describes "characteristics" of a development model (and the reason for each in parentheses):

- Small variation in module quality (same team);
- High quality modules (lead author);
- Less risk (team);
- Reasonable cost (resource people used only as needed);
- Collection of expertise (collective skills among team and resource people);
- Clear direction (team);
- Structured development and review process (team);
- More identification with project (lead author);
- Focus on content and pedagogy issues, not technical details (resource people);
- Minimal design-develop-assess cycle to reduce design investment and encourage exploration (personnel at one site), and
- Frequent meetings to share ideas and avoid pitfalls.

⁷³ A. Bork, in: Walker & Hess (Eds.), 1984, pp. 96 - 115

⁷⁴ M.L. Mitchell, A Minimum System for Development of Computer Assisted Instruction in Chemistry, Diss. (Kansas State University), Manhattan (Kans.), 1982, pp. 9, 65 - 66 [UMI No. 83 04338]

⁷⁵ Smith illustrates these factors in a proposal for the N.S.F.

Establishment of a taxonomy of development characteristics

The following taxonomy of development characteristics is the result of an adaptation and integration of the literature review and personal findings. The development characteristics are classified in two main groups:

- Characteristics due to the production approach, and
- Characteristics due to the production strategy.

Within this two main groups there can be found a set of sub-groups and characteristics as follows:

Characteristics due to the production approach:

- Integration of instructional knowledge into the development process (Design stages):
 - a) Focus on pedagogical issues (degree);
 - b) Timeline for the beginning of programming (after the design of the pedagogical issues);
 - c) Degree of program design determination by educational objectives;
 - d) Focus on content and pedagogy issues, not technical details;
 - e) Degree of following software quality standards and guidelines, i.e. instructional factors; and
 - f) Use of design tools.
- Implementation: Transfer of the design ideas and concepts:
 - a) Availability and establishment of document(s); and
 - b) Programming environment.
- Testing: Evaluation of the implementation:
 - a) Methods to assure the product is effective.
- Maintenance aspect
 - a) Easily revisable units;
 - b) Consistency across modules (common author); and
 - c) Maintenance loop as (major) part of software development.

Characteristics due to the production strategy:

- Staff involved in the development process:
 - a) Involvement of staff with appropriate skills, i.e. SME who focus on the interactivity of teaching contents;
 - b) Collection of expertise (collective skills among team and resource people);
 - c) Commitment of staff involved;
 - d) Number of staff involved for a single project; and
 - e) Specialization degree of staff.
- Financial aspect:
 - a) Reasonable costs (resource people used only as needed); and
 - b) Estimatable costs.
- Scale aspect of development:
 - a) Minimize transferability problems;
 - b) Overall quantity of single projects to manage at the same time; and

- c) Adaptability of the development approach to large-scale production (Strategies which may work for small amounts of production may be quite inadequate for large-scale production).
- Communication structure:
 - a) Between staff involved (frequent);
 - b) Role of documentation for communication;
 - c) Minimal design-develop-assess cycle to reduce design investment and encourage exploration (personnel at one site); and
 - d) Elapsed time for project steps.
- Project organization/responsibility and outcomes:
 - a) Totally SME/faculty-centered versus totally centralized;
 - b) (Clear) direction in development; (structured) development and review process; and
 - c) (Small) variation in module quality.

Discussion of development characteristics

It is very interesting that there could not be found much literature on development characteristics, though this may be a helpful instrument to identify (quality) enhancement factors and to describe and classify production systems. Though the term "development characteristics" could not be found in literature, we did identify some synonyms which seem to describe the same content. Bork applies the term "properties", Mitchell made "comments on development techniques" and Smith just called her indications "characteristics of a production model." Other synonymous terms are "procedural guidelines", and "procedural informations".

These development characteristics were put in a morphological grid, classified and completed on the level of the subgroups. The result of this procedure was the distinction of development characteristics due to the production approach, and characteristics due to the production strategy. Within this two main groups a set of subgroups has been established:

- Integration of instructional knowledge into the development process (Design stages);
- Implementation: Transfer of the design ideas and concepts;
- Testing: Evaluation of the implementation;
- Maintenance aspects;
- Staff involved in the development process;
- Financial aspects;
- Scale aspect of development;
- Communication structure; and
- Project organization/responsibility and outcomes.

We are convinced that the characteristics in these subgroups will be a helpful instrument to identify (quality) enhancement factors for the development of courseware because the concept of instructional system design (ISD) - including the heavy attention on the instructional/pedagogical design - and the software life cycle concept are represented in the taxonomy above. Moreover, questions of project organization, communication and management are addressed, too.

Conclusion

For we assume that it is easier to identify (quality) enhancement factors if there can be established a taxonomy of development characteristics which may guide the search for enhancement factors, a literature review was conducted first. But this review revealed that there is not much literature and the term "development characteristics" could not be found. Nevertheless we could establish a basic list of such criteria helpful for our further work to identify (quality) enhancement factors for the development of courseware.

3.3.2. CLASSIFICATION(S) OF COURSEWARE DEVELOPMENT APPROACHES AND STRATEGIES

Introduction

In the first section of this subchapter development characteristics have been identified and discussed. The next step will focus on the classification of courseware development approaches and strategies based on development characteristics. As it is not our primary concern to establish very detailed taxonomies of production system classifications, we just will give two (simple) examples. Our primary concern will be the identification of (quality) enhancement factors in the next subchapter.

It is necessary to identify criteria for a classification and taxonomy of production systems. We believe that the list of development characteristics identified is a powerful criteria set to establish such classifications because development characteristics, by definition, are forming parts and special qualities belonging to production systems which make it possible to describe and classify production approaches and strategies.

Given the taxonomy of development characteristics as criteria set to establish a classification of production systems, another problem emerges. The list of development characteristics encompasses more than 30 criteria. As a matter of fact, a taxonomy of production systems would require a comparison of more than 30 dimensions! But frequently discussed (whenever) are only a few of the total set of (possible) criteria:

- Individualized versus team development (number of staff involved and/or specialization degree of staff involved);
- Focus and integration of pedagogical and instructional issues versus technical focus; and
- Coding in all purpose (high level) language(s) (HLL) versus authoring languages/systems (AL/AS).

Selection of criteria to classify development approaches and strategies

The focus and integration of pedagogical and instructional issues versus technical focus is not discussed in this context because this criteria correlates with the question of individualized versus team development for which a classification will be established. Whenever a single person develops courseware totally alone, he/she will be a subject matter expert (SME) or a computer scientist but not both, normally. As a matter of

fact, the following development characteristics will be used as criteria to classify production systems:

- Individualized vs. team development (number of staff involved and/or specialization degree of staff involved);
- Implementation language (HLL vs. AS)

Two examples of production system classifications

A first example will be established for the criteria "specialization degree of staff involved" [low = (nearly) individual, medium = small team, high = detailed team strategy]. The classification is based (on soft data) of the description given in the chapter "overview of approaches and strategies".

a) low:

- Howard University: Faculty driven courseware development.

b) medium

- Academia's basic module: A faculty member working with a (student) programmer - a stand alone approach;
- The Courseware Development Group (CDG) at Dartmouth College;
- The informal small team approach at the University of Texas (Austin); and
- Stanford University: normal FAD projects.

c) high:

- CEEDE: Publishing for grades K through 12;
- CIDE instructional program development;
- The collaboration model;
- CONDUIT - Towards a service model;
- ETC/UCI: A team approach for large scale CBL material development;
- Stanford University: for special FAD projects; and
- Support model at the University of North Carolina at Chapel Hill (UNC-CH) (especially phase 2).

d) exterior to classification:

- CDEC (at CMU) working with CMU Tutor

A second example will be given based on the criteria "implementation language".

a) Implementation in all purpose language:

- CEEDE: BASIC, in future perhaps PASCAL
- CIDE: Depends on customer but a HLL is used normally.
- CONDUIT: Depends on author; investigation of C library
- CDG at Dartmouth College: PASCAL
- ETC/UCI: PASCAL
- University of Texas (Austin): More than 50 % in BASIC
- support model at UNC-CH: TURBO PASCAL (standard in phase 2)

b) Authoring languages and systems:

- CDEC at CMU: CMU Tutor
- Dartmouth College: An authoring system is in evaluation for faculty use
- Howard University: CMU Tutor
- University of Texas (Austin) Some projects are coded in PILOT
- Stanford University: Idea of CAT project
- UNC-CH: A few of the first projects (PC Pilot)

c) Unknown implementation environment:

- The collaboration model
- Stanford University: FAD projects

Discussion of classification examples

The first example given with the criteria "specialization degree of staff involved" reveals that most institutions visited use some sort of team strategy (medium or high) to develop courseware materials. For the time being, it is just Howard University which made a commitment for a faculty centered individual strategy to develop its first courseware projects. But even in this case the instrument of a weekly meeting was established to communicate and discuss project progress and problems with other developers. At Carnegie Mellon's Center for Design of Educational Computing (CDEC) individual and team based strategies are employed depending on the project. A distinction is made between low scale projects (such as demonstration material(s) for local use) which can be done with an individualized strategy and large scale projects where a team strategy is employed.

It is very interesting to see that nearly all institutions which were classified "high" work with subject matter experts/faculty exterior to the (professional) institution which develops (and publishes) the courseware! This is true for CEEDE, CIDE, and CONDUIT on a 100 percent base as all courseware is developed in conjunction with an outside subject matter expert. The inner-university institutions (as Stanford, Chapel Hill) work with faculty which are called in for a special project with the (professional) development group. This is true for the Courseware Development Group (CDG) at Dartmouth College, too. In cases where faculty members have more than one function and are responsible for the project's progress itself, the project management and overall progress of the project can become problematic. As a matter of fact, the progress depends in this case very heavily on the faculty member who has multi-functional responsibilities.

Finally, a trend can be seen that the number of institutions working with authoring tools and systems seems to increase. Stanford University initiated its Courseware Authoring Tool (CAT) project which intends to deliver faculty with authoring tools to develop courseware (nearly) on their own. Dartmouth College evaluates an authoring system for faculty use, too. At other institutions, there can be found ideas which guide in the same direction.

The second example supports this trend as, for the time being, most institutions develop their courseware in an all purpose (high level) language (PASCAL and some use BASIC). But at some institutions authoring languages were used (especially) in the beginning when first courseware projects were realized (e.g. at Chapel Hill). Moreover some institutions (still working with HLL) have ideas in mind to give faculty control of the process back, that is give faculty authoring tools. Interesting to see that publishers like CEEDE, CONDUIT (and CIDE) do not use authoring systems, for the time being, and do not intend to use such authoring tools in the foreseeable future. The Educational Technology at UCI belongs to this group, too.

Conclusion

A few criteria from the taxonomy of development characteristics were selected to establish two examples of production system classifications. It has been shown that the discussion "individualized versus team strategy" is only one aspect and criteria to classify production systems. The taxonomy would allow to establish such classifications with more than 30 dimensions!

The two examples revealed that many institutions work with subject matter experts/content specialists outside the (professional) development group. For the time being, most institutions use all purpose (high level) languages to code their courseware but there is an increasing number of institutions which works with authoring tools and systems. A reason for this trend is the idea to give faculty control of the overall process back and let them do what they want.

3.3.3. SUMMARY AND CONCLUSIONS

In this subchapter development characteristics were identified and two classification examples of production systems were presented. The development characteristics were identified from a literature review and findings from the trip in the U.S.A. As development characteristics are a helpful instrument to describe and classify production systems, we tried to establish two examples of such a classification of production systems. It could be shown that the "traditional" discussion of production system classification only encompasses a few of the development characteristics identified in the first section of this subchapter. But different classification schemes can be established based on a study's needs and the criteria employed.

3.4. (QUALITY) ENHANCEMENT FACTORS FOR COURSEWARE DEVELOPMENT (FOCUSING ON THE INTEGRATION OF THE INSTRUCTIONAL KNOW-HOW)

This third chapter has given a description of various and different courseware production approaches and strategies, and development characteristics were identified to classify production systems thus far. But the question remains: Are there (quality) enhancement factors for courseware development which (can) discriminate between good and bad courseware? This chapter focuses on this question with the aim to identify a list of such requirements.

The starting point is the identification of strengths and weaknesses in instructional system development and software system developed which permit to identify - in conjunction with the soft data gathered from the informative trip and from the literature review - a list of (quality) enhancement factors for courseware development.

3.4.1. IDENTIFICATION OF STRENGTHS AND WEAKNESSES IN COURSEWARE DEVELOPMENT

Problems encountered in instructional system development and use

During a conference workshop Romiszowski⁷⁶ and 30 instructional designers identified a matrix of problems encountered in instructional system development and use for small, medium and large-scale projects. The project scale is defined by the following criteria: Number of instructional designers on the project [1 to 5, 10 to 20, about 50], number of "client" institutions in the project [1 to 5, 10 to 20, about 50], and the number of students ("consumers") per year using the instructional system [about 100, 1000, 5000]:

	Stages in which problems are encountered		
Key problems when the project is:	Instructional design/production stages	Implementation/dissemination stages	Long-term management and control
Small	Promoting a creative approach	Availability of resources	Integration within the institution
	Adapting project to existing skills	Personnel changes and personalities	Performing long-term evaluation
Medium	Standardizing the language and documents	Creating "receptive" environment	Maintaining individualization
	Scheduling time and budgeting	Controlling "drift" in the system	Staff development, control, delegation

⁷⁶ For details see A.J. Romiszowski, 1986, pp. 381 - 382

	Stages in which problems are encountered		
Key problems when the project is:	Instructional design/production stages	Implementation/dissemination stages	Long-term management and control
Large	Coordination of several teams	Support of the "supra-system"	Management information systems that work
	Task scheduling and teamwork	Ensuring long-term resources	Support-staff requirements
	Planning regular validation and learning opportunities (for designers)	Controlling waste of resources/time	Student opinions, objectives, welfare
		Ensuring the system is self-correcting	Systematic procedures of project management
Factors which create problems in projects of any size but which increase as the project grows	Changes in the team or in the team management	Training of the teachers/monitors	Overcoming tendencies towards inertia
	Duration of the design stages and continuity in time	Extent of centralized control	Establishing effective command-chains
	Sheer quantity of material to be produced	Maintenance of interest/momentum	Predicting and defining future changes in system
		Ensuring adequate installation budget	Ensuring adequate running resources

This matrix of problems encountered in instructional system development and use seems to be a good representation of problems which can occur in such instructional system design projects. We are especially interested in the problems of the phase "Instructional Design/Production".

Problems encountered in software system development and use
Thayer et. al. identified 20 problems in software engineering project management (by literature reviews), restated them as hypotheses, and validated them through a widespread survey of 294 individuals (who returned the survey).²⁷

²⁷ R. H. Thayer, A. B. Pyster, and R. C. Wood, Major Issues in Software Engineering Project Management, in: IEEE Transactions on Software Engineering, Vol. SE-7 (1981), No. 4, pp. 333 - 342

Among the confirmed 13 problem issues (out of the 20) are:

- Requirement specifications are frequently incomplete, ambiguous, inconsistent, and/or unmeasurable;
- Success criteria for a software development are frequently inappropriate which results in poor "quality" delivered software, i.e. not maintainable, unreliable, difficult to use, relatively undocumented, etc.;
- Planning for software engineering projects is generally poor;
- Decision rules for use in selecting the correct software design techniques, equipment, and aids to be used in designing software in a software engineering project are not available; and
- Measurements or indexes of reliability that can be used as an element of software design are not available and there is no way to predict software failure, i.e. there is no practical way to show the delivered software meets a given reliability criteria.

The respondents were asked in which domain they would or did see a solution (managerial and/or technical)? Out of the 13 aforementioned definite problem issues, three were believed to be solved by improved project management, one by an improved technical solution and nine by both - improved project management and technical solutions.

Again another part of the questionnaire asked respondents how they would or did solve the stated problems. More than 4500 individual answers were given by the 294 people! Unfortunately, all of the responses are quite vague! The authors tried to categorize the answers and established a synopsis of 26 categories in which all the answers could be placed.

Among the 26 categories can be found:

- Use or enforce (existing) standards, procedures and documentation;
- Improve or insure requirement specifications to include ...;
- Analyze data from prior software development to determine best method of ...;
- Involve user or increase communication between user and developer in ...;
- Use good descriptive documentation techniques; and
- Define or consider quality in terms of deliverables, i.e., reliability, maintainability, etc.

To conclude: One of the interesting findings of the survey is that, at least without presenting a list of candidate solutions to software engineering personnel, they did not agree on how to solve major problems! Nevertheless, it was possible to categorize the vast amount of possible solutions suggested by the respondents in 26 categories.

A list of ideas, problems and recommendations made for courseware development

Throughout the trip, a lot of discussions did take place on ideas, problems encountered, and recommendations for the courseware development. This sections presents some of these ideas, problems, and recommendations made at the various sites visited:

- The design never can be perfect. After the coding there is an enormous (iterative) evaluation and revision needed to assure good quality of instructional software: Only the (summative) evaluation shows how students really work.
- The design and coding stages are very crucial. It is important to share ideas. There must be interactive work and good communication between the programmer and the faculty member during these phases to make progress.
- The application of academia's basic module (faculty member works together with student programmer) as courseware development strategy (for large scale material) is risky in the sense that much courseware being developed for education incorporates subject matter expertise with technical programming excellence, but tends to lack pedagogical and graphical/visual strength. This development is not without reasons. The omission of a pedagogical and graphics expert is economical. It simplifies the design process by removing a set of considerations which are important to the quality, but not to the existence of a product. Further it can be a difficult task to improve what a faculty member thinks is best.
- Another problem (at some university sites) is the local use of courseware and the weak interest to submit courseware for publication: The "teach it my own way" syndrome. For decades, faculty members in charge of a course have designed or redesigned the course to teach it in the way they felt it would be best taught. Though this has some virtues, there are negative aspects, too. One of these aspects to be aware of: the money spent for the development of courseware cannot be amortized over a series of years - and a large number of students! In conjunction with this problem comes the maintenance problem if courseware is developed by faculty on their own and there is not a commitment to general software engineering principles.
- At sites, working with a central group (with professional staff), some faculty members (= subject matter experts) had the idea in mind that the central group should support faculty more in the design and not limit faculty. Other faculty believed: "We thought they develop it, we have only to tell them what they have to do but it seems to be the other way round. They [= central group] tell us how we have to do things [e.g. the design]." Finally a faculty member of another site explained the pitfalls of an inadequate communication (there was no feedback and communication for weeks and/or even months; he was in doubt what is going on) with the central group. In other words: If the responsibility and communication structure is not clear, serious misunderstandings can emerge over time.
- As the point at first is to generate new ways of thinking about the concept and problem at hand, the total group shall sit together (including programmers) to make the user interface design and to help everyone get their minds open minded. Brainstorm, innovate, bounce around ideas is the motto.

⁷⁰ J.F. Rockart, M.S. Scott Morton, Computers and the Learning Process in Higher Education, New York, 1975, pp. 239 - 240

At another site they agree with this idea and procedure but do not include programmers during the design stage to make sure that the subject matter experts (teachers, faculty members) are really free in their ideas. Consequently, the design documents have to be put forward and backward until all features are clarified (programmable). This takes time, too!

- One group made the experience that factors which increase project cost(s) are inexperienced, slow, or just plain incompetent designer/programmers (probably the major factor); working on an authoring tool versus a smaller program; poor project definition or lack of faculty commitment; complex user interaction; the use of more than one programmer sequentially.
- Another idea focused on the establishment of design checkpoints (a sort of milestones) into the implementation schedule for a project, so that work can be redirected easily. At a checkpoint, the program has to be robust, clean, documented, and usable, but it may lack desired features.
- At one site, overall problems identified are the expense and time of development, the difficulty of prototyping quickly, and the generally weak programming environments available for large scale courseware development.

Conclusion

Problems encountered in instructional and software system development and use have been presented. These findings were supplemented by a list of ideas, problems and recommendations made by representatives of institutions visited in Spring 87.

3.4.2. IDENTIFICATION OF (QUALITY) ENHANCEMENT FACTORS FOR COURSEWARE DEVELOPMENT

Introduction

After having identified development characteristics (in conjunction with example classifications of production systems), strengths and weaknesses of instructional and software system development and use (including a list of ideas, problems and recommendations for courseware development), a list of quality enhancement factors will be presented in this section.

What we have produced here is an approach to condense and identify those quality enhancement factors which could make a difference in courseware development and improve quality development of courseware. These findings will need feedback, revision and modification as we learn more, but it is a START that we have found very useful in guiding our own thinking.

Review of development characteristic subgroups

The development characteristics identified in the preceding subchapter are not an aid to classify production systems only, but a guidance for identification of quality enhancement factors, too. The following subgroups of development characteristics have been identified for the purpose at hand:

a) Characteristics due to the production approach:

- Integration of instructional knowledge into the development process (Design stages);
- Implementation: Transfer of the design ideas and concepts;
- Testing: Evaluation of the implementation; and
- Maintenance aspects.

b) Characteristics due to the production strategy:

- Staff involved in the development process;
- Financial aspects;
- Scale aspect of development;
- Communication structure; and
- Project organization/responsibility and outcomes.

We directed our search of quality enhancement factors after these subgroups of development characteristics. We are convinced that the following subgroups are very important for courseware development:

- Integration of instructional knowledge into the development process (Design stages);
- Implementation: Transfer of design ideas and concepts;
- Testing: Evaluation of the implementation;
- Staff involved; and
- Communication structure.

We believe that the subgroups "Maintenance aspects", "Financial aspects", and "Project organization/responsibility and outcomes" will not be different from "traditional software development". As courseware is software, it underlies the same software engineering principles and methods as "traditional software for the maintenance issue. The financial aspects were not considered (see limitations presented in the introductory chapter). For the subgroup "Project organization/responsibility" (focusing on the issue of SME/faculty-centered versus totally centralized project responsibility, direction in development and review process, and variation in module quality) no consistent factor could be identified. Finally all the quality factors should be valid for the large scale development of courseware (but may be valid for the development of small scale developments, too).

Identification of factors

The following list of quality enhancement factors is the result of discussions, interviews and investigations at various courseware production sites in the U.S.A.

We distinguish between MUSTS and factors which are NICE but not necessary. The MUST factors are classified within the development characteristic subgroups. All the factors are seen to be disjunctive. It is not necessary that all the factors must be implemented in each production system developing quality courseware. It is rather the case that already a few of the factors may contribute and have a strong impact on quality courseware production. It would be very interesting to validate these factors in an empiric survey with other (large scale) courseware developers - not visited and investigated in Spring 87 - to assess the discrimination power of the list of quality enhancement factors.

Disjunctive MUSTS:

- Integration of instructional knowledge into the development process (Design stages)
 - 1) Whatever you do related to computers, think independent first. That means independent of the implementation aspect
—> Go as far as possible in initial design;
 - 2) Work with subject matter expert(s) (SME) which are exterior to the development group; and
 - 3) Apply an iterative process, i.e. iterative reviews and tests [after each development step: Developmental testing (= review after the design), Alpha testing (at the end of each phase = content review by a master performer), and Beta testing (validation of instructional effectiveness of the completed product)].
- Implementation: Transfer of the design ideas and concepts
 - 4) Use prototyping facilities; that is an implementation environment which supports the design activities (—> e.g. DEMO from Dan Bricklin, etc.).
- Testing: Evaluation of the implementation
 - 5) Make evaluations and field tests [Guidelines and standards can help and support this activity.]
- Staff involved in the development process
 - 6) Assure heart and mind commitment of staff involved (affective aspects); and
 - 7) Apply (some sort of) team strategy or at least collaboration with good/excellent content specialists/SME, instructional designers, and so forth.
- Communication structure
 - 8) Try to establish close and good communication between (team) members/staff. Possible aids to support this communication may be standard design documents and procedures, and establishing a standard schedule for the team members' meetings .

Disjunctive factors which are NICE but not necessary:

- 1) Programmer knowledgeable of the subject matter; and
- 2) Experience is helpful.

Discussion of factors

The list of quality enhancement factors presented tries to include all the sources available (preliminary findings from discussions, interviews, and the literature review). These preliminary findings were presented in the sections "Problems encountered in instructional system development and use", "Problems encountered in software system development and use", and "A list of ideas, problems and recommendations made for courseware development.

Some of the quality enhancement factors are accepted and well known in software engineering - such as the prototyping facilities - but it is important to know (and to be aware) that this general finding is valid for courseware development to the same extent or even more. Other factors are really specific to courseware development - e.g. to have the subject matter expert(s) exterior to the development group.

Taylor⁹⁹ focused in her study on the question whether the instructional system design process and its tasks can make a difference between good and bad courseware. One finding of the empiric study was that there is not a clear difference between publisher groups of low and high quality courseware in (instructional design) subtask compliance ratings.

Another finding of the same study revealed that the attitudes differentiate the courseware publisher groups very well. The group with low quality courseware seem to be less aware of development and design processes, considering them unimportant or irrelevant. Some of these publishers reacted negatively to these processes. The high quality courseware publisher group did not express such negative views of the development process.

Finally, Taylor could show that the most notable difference between the two groups (of courseware publishers) is in the production strategy. Publishers of high quality courseware generally use external subject matter experts (SME)/content specialists for much of their courseware development (for 75 % of published software), whereas the low quality group tends to do their courseware development in-house (for 84 % of published software).

Taylor's findings support the requirements two (Use SME exterior of the development group) and six (Ensure heart and mind commitment of staff involved).

Heines¹⁰⁰ establishes the following postulates in an article about authoring productivity:

- There is no substitute for time spent on design;
- Authoring languages and systems generally address the programming aspect of courseware development, not the design; and
- The production of quality courseware requires materials to be developed, tested, and revised in an iterative manner. The key of increasing productivity, then, is to attack the iterative process of development, testing, and revision.

Shneiderman¹⁰¹ emphasizes that ...

... iterative design methods that allow early testing of prototypes, revisions based on feedback from users, and incremental refinements suggested by test administrators are all necessary to arrive at a successful system.

These explanations support the requirement number three to apply an iterative process is very crucial, i.e. iterative reviews and tests [after each development step: Developmental testing (= review after the design), Alpha testing (at the end of each phase = content review by a master performer) and Beta testing (validation of effectiveness of the completed product)], and number one that the time spent on the (initial) design is very crucial and should be in a "free atmosphere" independent from implementation aspects.

⁹⁹ R. Taylor, 1986, pp. 161 - 176

¹⁰⁰ J. M. Heines, in: 1986/87, pp. 8 - 9

¹⁰¹ B. Shneiderman, Designing the User Interface, Strategies for Effective Human-Computer Interaction, Reading (MA), 1987, p. 390

The importance of **prototyping** facilities is well established and known and does not need further argumentation. It is the goal of early pilot studies, **prototypes**, and acceptance tests to force as much of the development process as possible into the prerelease phase, when change is relatively easy and inexpensive.¹⁰²

The need for evaluation is addressed in quality enhancement factor number three of applying an iterative manner of the development process. Factor five states the importance of the **evaluation** in the sense of **field testing**. Clarke¹⁰³ identified a few principles in an empiric study which are basic to the design of high quality courseware. One of the principles identified is the **evidence of field testing** to discriminate between high and low quality courseware.

Even Mitchell who developed some small scale courseware to investigate the question what is a minimum, efficient development system admitted that a developer should not be totally independent. It is very helpful to involve, at least indirectly, one or more interested people who can "see the forest for the trees".¹⁰⁴ Heines establishes the postulate that anybody cannot do courseware development alone; it takes a number of people with a range of skills.¹⁰⁵ This supports the "must" requirement number seven of establishing a sort of team strategy for courseware development.

The team strategy requires another quality factor: Close and good communication between team members/staff (requirement number eight). This finding is supported by various authors, e.g. Ally¹⁰⁶ who discovered that one of the most important factors in the team approach was the interaction and communication among team members. Reed et. al.¹⁰⁷ supplement this finding by the indication that daily contact with team members and extensive documentation of the project's progress is a vital part of managing a courseware development project, most effectively carried out through electronic mail (if the team members are not working at the same site).

A factor which can have an impact on quality courseware development is the knowledge and experience of staff involved. Montague et. al. illustrate that it is not likely that **artistry** can ever be eliminated completely. The **expertise**, therefore, of the personnel developing courseware is and will be vitally important to its quality and effectiveness.¹⁰⁸

¹⁰² B. Shneiderman, 1987, p. 397

¹⁰³ F. Clarke, 1984, p. 131

¹⁰⁴ M. L. Mitchell, 1982, p. 64

¹⁰⁵ J. M. Heines, Anybody Can't Do CBT, in: Training News, 1985, March, p. 9

¹⁰⁶ M. Ally, A Team Approach to Computer Courseware Design, in: Educational Technology, Vol. 25 (1985), No. 7, pp. 28 - 30

¹⁰⁷ M. J. Reed et. al., Effective Management of CAI Projects, in: Proceedings of 26th ADCIS Conference, pp. 18 - 23

¹⁰⁸ W. E. Montague, et. al., Quality CBI Depends on Quality Instructional Design and Quality Implementation, in: Journal of Computer Based Instruction, Vol. 10 (1983), Nos. 3 & 4, pp. 90 - 93

Conclusion

The list of quality enhancement factors identified is a small contribution to the field of **production systems** and a **further step** in closing the gap between all the literature which is published and the overall quality of courseware as indications are given for practitioners which factors may discriminate between high and low quality courseware related to the production system.

The findings may not be transplanted directly to a production system in use without making any reflections and adequate operationalizations to make sure that **constraints** and **circumstances** of that specific production system may be adapted and included.¹⁰⁷

3.4.3. SUMMARY AND CONCLUSIONS

After discussing problems encountered in instructional system development and use, problems encountered in software system development and use, and a list of ideas, problems and recommendations made for courseware development, a list of quality enhancement factors was identified. The factors are classified in two subgroups: "MUST" requirements and factors which are "nice but not necessary". The list of quality factors is the result of analyzing soft data - discussions, interviews and findings from literature.

The findings are not seen as a perfect result, rather they will need **feedback**, **revision** and **modification** as we learn more, but it is a **start** that we have found very useful in guiding our own thinking.

3.5. SUMMARY

This chapter introduced twelve production systems of institutions which develop courseware in the U.S.A. The systems explained represent a fair and interesting sample of possible production systems for courseware development. Most institutions - especially the universities - see themselves still as **pioneers** in the field. It was interesting to realize that a few of the institutions change their center of interest from developing courseware to building courseware authoring tools.

As the question of working with an authoring language (AL) and/or system (AS) is very popular, the overall role of such AL/AS was investigated in a subsequent section. One of the findings related to AL/AS was that these tools (try to) reduce the programming complexity, but still require the author to provide the instructional design. AL/AS, therefore, can be an appropriate production environment depending on the staff who has to develop which kind of courseware under which circumstances and constraints.

¹⁰⁷ For details see introductory chapter (footnote 16, pp. 7 - 8).

The next step was the identification of development characteristics. Though the literature review did not reveal much material, it was possible to establish a taxonomy of such development characteristics which were used to establish example classifications of production systems.

Another function of the development characteristics is their guidance in identifying quality enhancement factors. These factors were derived from soft data - discussions, interviews, a literature review, and a review of production systems. These factors and/or requirements (can) make a difference in quality courseware development and may be adapted in existing production systems.

4. SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

Summary

The starting point of this report is the well-known fact that the overall quality of courseware is bad. A bundle of reasons contribute to and may explain this situation. We believe that the production system (which includes the production strategy and the production approach) of courseware is one of the (major) reasons to explain the situation. Moreover, we are of the opinion that the production system has a strong impact on quality courseware development.

A research study was established therefore to investigate the question of (large scale) courseware quality with the aim to identify such quality enhancement factors which (can) make a difference between high and low quality courseware in relation to the production system.

The literature review on production systems (approaches and strategies) revealed that it would prove very helpful - even necessary - to study such production systems in reality. It was found that a good sample of production systems could be explored in the U.S.A. as there is much development and many activities going on in courseware development. This trip has taken place in Spring 87 (from end of February through Easter).

Chapter two of this report gave an introduction into the background of this report - the relationship(s) between instructional (system) design and software engineering. It has been shown that the ISD process and the software life cycle have many concepts and ideas in common!

This ambiguity of courseware development requires that production systems are established specific to courseware development. A production system has to integrate ideas of instructional design, therefore, and respect the preliminaries of software engineering on the other hand. This makes courseware development a specific task which has to be distinguished from "traditional" instructional material design (i.e. books) and "traditional" software development!

The main chapter of this report focused on production approaches and strategies for courseware development. It encompassed a description of twelve production systems. As the question of working with an authoring language (AL) and/or system (AS) is very popular, the overall role of such AL/AS was investigated in a subsequent section. One of the findings related to AL/AS is that these tools (try to) reduce the programming complexity, but still require the author to provide the instructional design. AL/AS, therefore, can be an appropriate production environment depending on the staff who has to develop which kind of courseware under which circumstances and constraints.

Development characteristics of production systems were condensed from a literature review, discussions, interviews, and can be employed to classify production systems. It has been shown that the traditional discussion in this area (e.g. authoring systems versus all purpose (high level) languages or individual versus team approach) only covers a small part of characteristics which can be used to describe and classify production systems. As a matter of

fact, two examples of such production system classifications have been presented after reducing the number of development characteristic subgroups and criteria employed for the classifications.

These development characteristics were the starting point for the establishment of (quality) enhancement factors which were identified from literature, a review of production systems, discussions, and interviews. Though some of the factors identified are well accepted (especially in software engineering), others are quite new and not well-known.

The factors have been classified in two subgroups: "MUST" requirements and factors which are "NICE" but not necessary. The findings are not seen as a perfect result, rather they will need feedback, revision and modification as we learn more, but it is a start that we have found very useful in guiding our own thinking.

Conclusions and Recommendations

We are convinced, to draw a personal conclusion first, that it was worth the time and the money to spend two months in the U.S.A. in order to investigate the question of quality courseware development.

Another conclusion focuses on the existing wealth of production systems for courseware development. We are aware of the fact that there is not a single and/or unique production strategy and/or approach which could ensure good quality development of courseware alone. There are a wide range of systems which represent many years of experience, specific circumstances and constraints, and know-how. But we believe that the (quality) enhancement factors identified (can) discriminate between low and high quality of courseware, i.e. increase the likelihood of quality courseware development, and can be instructional for institutions which recently joined the area, and/or are planning to do so in the foreseeable future.

It would be interesting, therefore, to go a step ahead in the area of production systems in order to investigate development characteristics, classifications of production systems, and quality enhancement factors even more. It is recommended here to validate the development characteristics and quality enhancement factors, e.g. through a delphi process. Finally, we realized that terms such as instructional factors, development characteristics, and so forth are not used in a consistent way in literature. Many synonyms are existing and do not facilitate the comparison, description and investigation of related matters of interest.

Considering the situation in Switzerland, we believe that the (large scale) courseware development and use in most areas and levels of business and education will be in the trial and error stage for many years, if not a (few) set pattern(s) of development and usage will be established soon, e.g. in development centers for instructional software at various sites (in business and education). This seems to be the only way to gain experience and to go a step ahead as we are of the opinion that it is the combination of instructional creativity and experience of instructional computing/computer based training that can evolve new instructional materials (courseware) and push forward research, too.

5. LIST OF VISITED INSTITUTIONS IN THE U.S.A.

The following list gives an overview of all visited institutions, the main appointments, and the interviews made at all the various sites. All indications are given in alphabetic sequence.

a) California

01) Stanford University at Palo Alto

- Academic Computing and Information Systems (ACIS) and Instruction and Research Information Systems (IRIS)
Sweet Hall, 3rd Floor
Stanford (CA) 94305-3091
Tel: (415) 723 1055
- Michael Carter, Director IRIS
Sweet Hall, 3rd Floor
Tel: (415) 723 1055
- Decker Walker, Prof. School of Education
Cubberly Auditorium (2nd Floor)

02) University of California at Berkeley

- Graduate School of Education
Division of Education in Mathematics, Science and Technology
1600 Tolman Hall
Berkeley (CA)
Tel: (415) 643 6379
- Marcia C. Linn, Assistant Prof.
Graduate School of Education
Room 4611

03) University of California at Irvine

- University of California at Irvine
Educational Technology Center (ETC)
Irvine (CA) 92717
- Alfred Bork, Director ETC
Tel: (714) 856 6945

b) Georgia

04) 5th CBT Conference at Atlanta (March, 15 - 18)

c) Illinois

05) Computer Teaching Corporation at Champaign

- Computer Teaching Corporation (CTC)
Illni Plaza
1713 South Neil
Champaign (IL) 61820
Tel: (217) 352 6363
- David Stone, Ph.D.
Director of Software Applications

d) Iowa

06) CEEDE

- The University of Iowa
CEEDE
N345 Oakdale Hall
Oakdale Campus
Oakdale (IA) 52319
- Prof. L. M. Stolurow
Director CEEDE
Tel: (319) 335 4116

07) CONDUIT at Iowa City

- The University of Iowa
CONDUIT
Oakdale Campus
Iowa City (Iowa) 52242
Tel: (319) 335 4100
- Pete Trotter, Director CONDUIT
Tel: (319) 335 4100

e) Maryland

08) Center for Instructional Development and Evaluation (CIDE)

- University of Maryland University College
Center for Instructional Development and Evaluation (CIDE)
University Boulevard at Adelphi Road
College Park (MD) 20742
Tel: (301) 985 7122
- Gary A. Hettinger, Assistant Director
Evaluation and Research
Tel: (301) 985 7356

- Michael Hastings Spencer, Production Director
Tel: (301) 985 7356

09) Interactive Technology, Inc. at Laurel (Baltimore)

- Interface Technology, Inc.
Microcomputers and Robotics
14440 Cherry Lane Court
Suite 219
Laurel (MD) 20707
- Patrick H. Stakem, President
Tel: (301) 490 3608
or: (301) 953 1199

10) University of Maryland at Catonsville (Baltimore) (UMBC)

- University of Maryland
Department of Education
5401 Wilkens Avenue
Catonsville (MD) 21228
- Robin Taylor, Ph.D.
Coordinator, Computer Education
Department of Education, UMBC
Acad IV, Wing A, Room 432
Tel: (301) 455 2305
- James S. Vincent, Prof. of Chemistry
Chemistry Department
422 C/P Building
Univ. of Md (MD) Balto. County
- Debbie Hoover, Programmer
Fine Arts Department
539 FA Building
Univ. of Md (MD) Balto. County

f) Massachusetts

11) Computer Museum at Boston

- The Computer Museum
Museum Wharf
300 Congress Street
Boston (MA) 02210
Tel: (617) 426 2800

12) Harvard University at Cambridge

- Educational Technology Center
Graduate School of Education
Gutman Library, Room 337
Appian Way
Cambridge (MA) 02138
Tel: (617) 495 9373
- Prof. Judah Schwarz, Co-Director

g) New Hampshire

13) Dartmouth College at Hanover

- Dartmouth College
Hanover (N.H.) 03755
Tel: (603) 646 1110
- Jamshed Bharucha, Assistant Prof. of Psychology
403B Silsby Hall
- Richard Birnie, Associate Prof. of Earth Science
223 Fairchild Hall
- Steve Maker, Manager of Courseware Development
3D Webster Hall
- Robert Sokol, Prof. of Sociology
114 Silsby Hall
- Donald Spicer, Director of Academic Computing
Kiewit Computer Center
Tel: (603) 646 3241

h) North Carolina

14) University of North Carolina at Chapel Hill (UNC-CH)

- University of North Carolina (UNC) at Chapel Hill
Chapel Hill (NC) 27514
- Microcomputing Support Center (MSC)
UNC
402 Hanes Hall 019A
Chapel Hill (NC) 27514
- UNC/IBM Courseware Development Project (CDP)
4001 Davis Library 0801
The University of North Carolina
Chapel Hill (NC) 27514

- Molly Anderson, Project Coordinator
CDP
Davis Library
Tel: (919) 962 2036
- Mark I. Appelbaum, Prof. of Psychology/Statistics
358A Davie Hall 013A
Tel: (919) 962 5036
- Margret Hazen Ph.D. ,
Director of Instructional Computing
Microcomputing Support Center
Tel: (919) 962 1175
(= contact person over long term time frame)
- Marya Ilgen-Lieth, Instructional Designer
CDP
Davis Library
Tel: (919) 962 2036
- Joseph C. Lowman, Prof. of Psychology
Department of Psychology
257 Davie Hall 013A
Tel: (919) 962 5432
- Jay Nievergelt, Prof. of Computer Science
Computer Science Department
124 New West
Tel: (919) 962 7002
- Ralph Wileman, Prof of Education/Instructional Design
School of Education
06 Peabody Hall 037A
Tel: (919) 962 3291

i) Pennsylvania

15) Carnegie Mellon University at Pittsburgh

- Center for Design of Educational Computing (CDEC)
500 Forbes Avenue Building E
Pittsburgh (Pa) 15213-3890
- David Trowbridge, Research Scientist
CDEC
Tel: (412) 268 8527

j) Texas

16) University of Texas at Austin

- Instructional Computing
University of Texas
Austin (TX) 78712
Tel: (512) 471 4900
- Shanti Aggarwal, Prof. of Biomed. Engineering
612 Engineering Science Building
Tel: (512) 471 5367 or 7167
- George H. Culp, Ph.D.
Assistant Director of Instructional Computing
147 Taylor Hall
Tel: (512) 471 3241
- Norma Fowler, Prof of Botany
108B Biological Laboratories
Tel: (512) 471 1295 or 4546
- Joe J. Lagowski, Prof. of Chemistry
4.420 Welch Hall (Chemistry)
Tel: (512) 471 3288
- Randy Machemehl, Prof. of Civil Engineering
6.902 Cockrell Hall
Tel: (512) 471 4379

k) Washington, D.C.

17) Howard University at Washington, D.C.

- Roy Beasley, Prof., Architecture and Planning
- Lewis W. Carr, Prof., Social Work
- Jesse Paul Clay, A.M., Ph.D.
Academic Budget and Research Officer, Adjunct Prof. of
Mathematics
Office of the Vice President for Academic Affairs
2400 6th Street, N.W.
Washington, D.C. 20001
Tel: (202) 636 6055 or 6056
- Wanda Michener-Colston, Assistant Prof., Psychology
- Raymond Williams, Prof., Chemistry

1) Miscellaneous Contacts

18) Aimtech Corporation

77 Northeastern Boulevard
Nashua (NH) 03062

John Olapurath, Chairman
Tel: (603) 883 0220

19) Allen Communications

5225 Wiley Post Way
Salt Lake City (Utah) 84116
Tel: (801) 537 7800

20) American Educational Research Association

Central Office
1230 Seventeenth Street, N.W.
Washington, D.C. 20036
Tel: (202) 223 9485

21) Brown University at Providence (R.I.)

- Institute for Research in Information and Scholarship
(IRIS)
Brown University
Box 1946
155 George Street
Providence (RI) 02912
Tel: (401) 863 2001
- Sindi Terrien
IRIS Area Associate
Tel: (401) 863 3438

22) Columbus College at Columbus (GA)

Abbott Turner School of Business
Columbus (GA) 31993
Tel: (404) 568 2284

23) International Association for Computing in Education (IACE)

P.O. Box 57163
Washington, D.C. 20037
Tel: (612) 255 9440

24) Kinko's Service Corporation

4141 State Street, Suite B
Santa Barbara (CA) 93110-1891
Tel: Tollfree USA: 800 - 235 6919
(805) 967 01 92

25) National Society for Performance and Instruction

1126 Sixteenth Street, N.W., Suite 214
Washington, D.C. 20036
Tel: (202) 861 0777

6. LIST OF AVAILABLE DOCUMENTS

The following list gives an overview of all documents gathered throughout the trip in the U.S.A. All indications are given in **alphabetic** sequence.

a) California

01) Stanford University at Palo Alto

ACIS ¹¹⁰	ACIS - Academic Computing and Information Systems, Stanford University, January 1987
ACIS/IRIS ¹¹¹	Creating Courseware, Faculty Author Development Project, October 1986
ACIS/IRIS	Instruction and Research Information Systems, September 1986
ACIS/IRIS	IRIS, Consulting Services, October 1986
ACIS/IRIS	IRIS Documentation, October 1986
ACIS/IRIS	IRIS, Interactive Classroom Experiment, October 1986
ACIS/IRIS	Microcomputer Evaluation Lab, September, 1986
ACIS/IRIS	Microcomputer Clusters at Stanford University, January 1987
ACIS/IRIS	Speaking of Computers..., Issue No. 8, January 5, 1987
ACIS/IRIS	Students Authoring Software Project, Information and Guidelines, February 1987
ACIS/IRIS	Students Authoring Software Project, Information About Proposals and Awards, February 1987
ACIS/IRIS	Welcome to Computing at Stanford University, September 1986
IRIS	About Computing at Stanford, A Guide for Faculty and Students, September 1986
SMUG ¹¹²	Winter Calendar, Winter 1986-7

02) University of California at Berkeley

Division of Education in Mathematics, Science and Technology, Graduate School of Education	Graduate Programs at the University of California, Berkeley in Science Education, Cognitive Science, and Information Technology
Graduate School of Education Linn, Marcia C.	1986-87 Announcement Establishing a Research Base for Science Education: Challenges, Trends, and Recommendations, Report of a national conference held January 16-19, 1986

¹¹⁰ ACIS = Academic Computing and Information Systems

¹¹¹ IRIS = Information and Research Information Systems

¹¹² SMUG = Stanford Macintosh Users' Group

03) University of California at Irvine (UCI)

- | | |
|------------------------|--|
| Bork, Alfred | Computer-Based Learning Modules to Aid Problem Solving, March 10, 1987 |
| Bork, Alfred | Interaction, Lessons from Computer Based Learning, September 8, 1986 |
| Bork, Alfred | Let's Test The Power Of Interactive Technology, in: Educational Leadership, March 1986, pp. 36 - 37 |
| Bork, Alfred | Pedagogical Development of CBL Material, in: A. Bork and H. Weinstock (Eds.), Designing CBL Materials, NATO ASI Series, Vol. F 23, Berlin/Heidelberg, 1986, pp. 67 - 94 |
| Bork, Alfred | Planning for the Future of Education, April 7, 1987 |
| Bork, Alfred | Production Systems for Computer-Based Learning, in: Walker and Hess (Eds.), Instructional Software, Principles and Perspectives for Design and Use, Belmont (CA), 1984, pp. 96 - 115 |
| Bork, Alfred | Recent Books on Computer Based Learning |
| Bork, Alfred | The Potential for Interactive Technology, in: BYTE, February 1987, pp. 201 - 204 |
| Bork, Alfred | The Role of Computers in Mastery-Based Courses, July 16, 1984 |
| Bork, Alfred | UCI Computer Programs to Learn to Recognize Japanese, January 13, 1987 |
| ETC/UCI ¹¹³ | Activities, March 4, 1987 |
| ETC/UCI | Computer Dialogs Developed at Irvine, June 5, 1987 |
| ETC/UCI | Statement of Capabilities, March 2, 1987 |

b) Georgia

04) 5th CBT Conference & Exhibition in Atlanta

- | | |
|--------------------------------------|---|
| Weingarten Publications, Inc. (Eds.) | The 1987 CBT Guide, Computer-Based Training Systems and Courseware |
| Weingarten Publications, Inc. (Eds.) | The 5 th Computer-Based Training Conference & Exposition, Program Book and Exhibitor Directory |
| Weingarten Publications, Inc. (Eds.) | Training News, A Monthly Newspaper for the Training Professional, Vol. 8, No. 7 |

Session Handout Nos.:

101	CBT Standards
103	Politics of CBT Implementation
104	Designing Effective Interaction
106	Instructional Data Base
201	Chuckle Based Training
202	Introduction to Interactive Video
203	Hiring CBT Authors
204	The Dynamics of Help
205	Screen Design Techniques to Facilitate Different Learning Processes
206	The Design Document
207	Summing up Your CBT (Evaluation)
301	Evaluating Off-the-Shelf CBT
303	Practical Solutions to CBT Design Prob- lems
306	The Paper Chase: Your Blueprint for Electronic Creation
307	A Systems Approach to Courseware Devel- opment
401	Glitz vs. Guts: Design Extremes
402	Creating an Image with Graphics
403	Delivering CBT in the Large Organiza- tion
404	Intelligent CBT-Building Knowledge Bases
405	Success with Embedded Training
407	Toward Quality Control in CBT
502	Internal Design Review: The Structured Walkthrough Technique
504	Interactive Video: A Corporate Informa- tion Tool
505	Managing CBT in an Integrated Environ- ment
506	Automating Instructional Design
508	Avoiding Design Traps
602	"Playing isn't playing" (Game: Got'cha)
605	Creating Lessons with Shell Authoring Systems
702	The Information Center: A Home for CBT
703	CBT for Product Knowledge (Design)
708	Prototyping New Applications with CBT
804	Managing the CBT Environment
807	Moving CBT Development In-House: Tools for Organization & Productivity

Various Handouts and Documents from the Exhibition Focusing on
Authoring Languages and Systems

c) Illinois

- 05) Computer Teaching Corporation (Champaign)
No special documents gathered

d) Iowa

- 06) CEEDE
CEEDE

ESL/Bilingual Materials Catalog¹¹⁴

- 07) CONDUIT at Iowa City
CONDUIT

CONDUIT

Author Assist, Tools for Microcomputer Courseware Development, Apple II Version 0.2, 1985

CONDUIT

Catalog of Educational Software, Secondary - College Level, 1986-1987

CONDUIT

CONDUIT Educational Software Submission Questionnaire, September 1986

CONDUIT

DASHER, An Answer Processor for Language Study, User's Guide

CONDUIT

Notes on Documentation for CONDUIT Authors, September 1986

CONDUIT

Notes on Instructional Design for CONDUIT Authors, September 1986

CONDUIT

Notes on Programming for CONDUIT Authors, September 1986

CONDUIT

Various single sheets on CONDUIT, its organization, development processes, and so on

CONDUIT

Submission Guide for CONDUIT Educational Software, September 1986

Various DASHER Drill Handouts for various Languages

e) Maryland

- 08) Center for Instructional Development and Evaluation (CIDE)
CIDE

Innovative Solutions to Your Educational Challenge, "Small Issue"

CIDE

Innovative Solutions to Your Educational Challenge, "Large Issue"

- 09) Interactive Technology, Inc. at Laurel (Baltimore)
Interface Technology, Inc.

Handout encompassing various fields

- 10) University of Maryland at Catonsville (Baltimore) (UMBC)
Taylor, Robin

An Analysis of Development and Design Models for Microcomputer-Based Instruction, Diss. (Syracuse University), 1986

Taylor, Robin

Cost and Benefit Considerations in Educational Publishers' Decision-Making, Syracuse, June 1984

¹¹⁴ ESL = English as Second Language

Taylor, Robin

Various Paper Copies of Slides Focusing on Courseware Development Procedures and Results of an Empiric Study (related to the dissertation)

Taylor, Robin, and Ely,
Donald P.

Courseware Quality: Do Development and Design Procedures Make a Difference, in: EURIT '86, pp. 405 - 410

f) Massachusetts

11) Computer Museum at Boston

No special documents gathered.

12) Harvard University at Cambridge

ETC

Educational Technology Center, Second Year Report, November 1985

ETC

ETC Targets, Vol. 3, No. 1 (winter 1986)

ETC

ETC Targets, Vol. 3, No. 3 (fall 1986)

ETC

ETC Targets, Vol. 4, No. 1 (winter 1987)

g) New Hampshire

13) Dartmouth College at Hanover

ACM/SIGCUE

Outlook: Computing in Higher Education, Vol. 19 (1986), Nos. 1/2

Bharucha, J. J., Meike,
B., and Baird, J. C.

The Macintosh as a user-friendly laboratory for perception and cognition
Courseware Catalog, Version 4.0 - January 20, 1987

Dartmouth College

Stebbins, Marie, and
Allen, Margret
The MacCommittee

Computing at Dartmouth College, 1986
The Impact of the Macintosh at Dartmouth (1984/1985)

h) North Carolina

14) University of North Carolina at Chapel Hill (UNC-CH)

Anderson, Molly

Instructional Software Development: Procedures and Resources, June 1986

Anderson, Molly, and
Hazen, Margret

Types of Courseware, Drill & Practice, May 1985

CDP¹¹⁵

Descriptions of Courseware Projects, February 1987

CDP

Newsletter, October 1984

CDP

Newsletter, Vol. 1 (1984), No. 3 (December)

CDP

Various Handouts Focusing on Evaluation of Courseware

Graves, William H.	Anatomy of Courseware Development, November 1984
Graves, William H	Educational Software: One University's Perspective, March 1986
Hazen, Margret	Courseware Evaluation, May 1985
Hazen, Margret	Criteria for Choosing Among Instructional Software Authoring Tools
Hazen, Margret	Instructional Software Design Principles, in: Educational Technology, November 1985, pp. 18 - 23
Hazen, Margret	RTSID Presentation
Hazen, Margret	When to Use the Team Approach, Decision Making Handout, 28 th ADCIS Conference Presentation, November 1986
Hazen, Margret, and Anderson, Molly	Advantages of Computers over Other Instructional Media, October 1985
Ilgen-Lieth, Marya	Instructional Software Paths for Demonstration and Evaluation, June 1986
Ilgen-Lieth, Marya	Student Evaluations of CBIM, November 1985
MSC ¹¹⁶	CBIM = The Team Approach, September 1985
MSC	Instructional Software Development (ISD) News, Vol. 2 (1986), No. 3 (June)
MSC	Instructional Computing (IC) News, Vol. 2 (1986), No. 7 (October)
MSC	Instructional Computing (IC) News, Vol. 2 (1986), No. 8 (Nov./Dec.)
MSC	Instructional Computing (IC) News, Vol. 3 (1987), No. 1 (January)
MSC	Instructional Computing (IC) News, Vol. 3 (1987), No. 2 (February)
MSC	Instructional Computing (IC) News, Vol. 3 (1987), No. 3 (March)
MSC	Microcomputer User Service, January 1987
MSC	Microcomputing Support Center, Services, March 1987
MSC	Mission Statement, March 1986
MSC	The UNC/IBM Project, April 1986
MSC	Using the Labs, March 1987

i) Pennsylvania

15) Carnegie Mellon University at Pittsburgh

Academic Services Division

Larkin, Jill H., Trowbridge, David E., and Scheftic, Carol

Computing and Library Services of Carnegie Mellon University, November 1986

An Operational Tutor Combining Techniques of Intelligent and Traditional Computer-Assisted Instruction, Session paper, International Meeting on Intelligent Tutoring Systems, Tel-Aviv (Israel)

¹¹⁶ MSC = Microcomputing Support Center

Ludwig, T. J. (Ed.)	Carnegie Mellon - Accent on Research, No. 3, October 1986
Maguire, Matthew	Andrew Is Alive and Well and Living in the Computer Network, Spring 1986
Morris, J. H., et. al.	Andrew: A Distributed Personal Computing Environment, in: Communications of the ACM, Vol. 29 (1986), No. 3 (March), pp. 184 - 201
N.N.	Inter-University Consortium for Educational Computing
Sherwood, Bruce A.	Workstations at Carnegie Mellon, IEEE Computer Society Reprint, Reprinted from the Proceedings of the Fall Joint Computer Conference, November 1986
Sherwood, Bruce A., and Sherwood, Judith N.	CMU Tutor: An Integrated Programming Environment for Advanced-Function Workstations
Talukdar, Sarosh N., and Joobhani, Rostam	A Knowledge-Based Intelligent Tutor for Thevenin Equivalents, Session Handout
Trowbridge, David	A Sampler of Educational Software at CMU
Trowbridge, David	Computer-Based Tutor on Graphic Equations
Trowbridge, David	Quick Generation of Lecture Demonstrations and Student Exercises
Trowbridge, David	Using Andrew for Development of Educational Applications
University Relations (Eds.)	Distributed Personal Computing Environment, February 1986
University Relations (Eds.)	Educational Software Development, February 1986
University Relations (Eds.)	Educational Computing, February 1986

j) Texas

16) University of Texas at Austin (UTA)

N.N.	Informal Project Review
TLTG	TLTG - The Texas Learning Technology Group, January 1987
UTA	Progress Report Project QUEST, 12-15-1986
UTA	Progress Report Project QUEST, 01-07-1987

k) Washington, D.C.

17) Howard University at Washington, D.C.

Handout/Explanations of Howard University's Courseware Development Laboratory

1) Miscellaneous Contacts

18) Aimtech Corporation

No special documents gathered.

19) Allen Communications

No special documents gathered.

20) American Educational Research Association (AERA)

AERA 1987 AERA Annual Meeting Program,
Washington, DC, April 20 - 24

21) Brown University at Providence (R.I.)

Garrett, L. Nancy, and
Smith, Karen E.

Building a Timeline Editor from Prefab
Parts: The Architecture of an Object-
Oriented Application, in: OOPSLA 86
Proceedings, pp. 1 - 16

Garrett, L. Nancy, Smith,
Karen E., and Meyrowitz,
Norman

Intermedia: Issues, Strategies, and
Tactics in the Design of a Hypermedia
Document System, in: Conference on Com-
puter-Supported Cooperative Work, De-
cember 1986

Haan, Bernard J., Drucker,
Steven M., and Yankelo-
vich, Nicole

An Object-Oriented Approach to Develop-
ing Consistent, Integrated Applica-
tions, in: IRIS Technical Report, No.
3, 1986, pp. 1 - 11

IRIS¹¹⁷

IRIS

Meyrowitz, Norman

Hypermedia Bibliography, March 1987

Intermedia, February 1987

Intermedia: The Architecture and Con-
struction of an Object-Oriented Hyper-
media System and Applications Frame-
work, in: OOPSLA 86 Proceedings, Sep-
tember 1986, pp. 186 - 201

N.N.

The Office of Program Analysis, March
1985

Shields, Mark

Computing at Brown University, A Survey
Research Report Summary, October 1984

Shipp, William, Michel,
Martin J., and Beeman,
William O.

The Institute for Research in Informa-
tion and Scholarship (IRIS), November
1986

Yankelovich, Nicole

INTERMEDIA: A System for Linking Multi-
media Documents, in: IRIS Technical Re-
port, No. 2, 1986, pp. 1 - 18

Yankelovich, Nicole, and
van Dam, Andries

Spinning Scholarly Webs, Essay for
Annenberg's Report to Educators

Yankelovich, Nicole,
Meyrowitz, Norman, and
van Dam, Andries

Reading and Writing the Electronic
Book, in: Computer, October 1985, pp.
15 - 30

22) Columbus College at Columbus (GA)
No special documents gathered.

23) International Association for Computing in Education (IACE)
No special documents gathered.

24) Kinko's Service Corporation
Kinko's Copies Academic Courseware Exchange Catalog,
Fall 1986
Kinko's Copies Academic Courseware Exchange Catalog,
Spring 1987

25) National Society for Performance and Instruction (NSPI)
NSPI Improving Human Performance, 25th Anni-
versary 1987 Conference Program, March
16 - 21
NSPI Presenter's Directory, Improving Human
Performance, 15th Anniversary, 1987

m) Miscellaneous Documents

Collis, Betty, and Gore
Marilyn

The Collaborative Model for Instruc-
tional Software Development, in: Edu-
cational Technology, February 1987, pp.
40 - 44

DEC

edu Magazine, Issue Number 44, Spring
87

Hewett, T. T

When Every Student Has a Computer: A
New Perspective on Courseware and Its
Development, in: Behavior Research
Methods, Instruments, & Computers, Vol.
18 (1986), pp. 188 - 195

Kearsley, Greg

33 Ways to Better Software Design, in:
Training and Development Journal, July
1986, pp. 47 - 48

Merrill, David M.

Prescriptions for an Authoring System,
in: Journal of Computer-Based Instruc-
tion, Vol. 14, No. 1, pp. 1 - 10

Naiman, Adeline

A Hard Look at Educational Software,
in: BYTE, February 1987, pp. 193 - 200

Osgood, Donna

The Difference in Higher Education, in:
BYTE, February 1987, pp. 165 - 178

Perkey, M. N.

The Effect of a Machine-rich Environ-
ment on Courseware Development: The
Process and the Product, in: Behavior
Research Methods, Instruments, & Com-
puters, Vol. 18 (1986), pp. 196 - 204

7. REFERENCES

- ACIS/IRIS
Creating Courseware, Faculty Author Development Project, Stanford University, October 1986
- ADCIS
28th International Conference Proceedings, Bellingham (ADCIS: Western Washington University), 1986
- Adler, Karen
The Computerization of Higher Education, The Experience of Pioneers in the Field, ETH Report No. 86-07, Zürich, 1986
- Alessi, Stephen M., and Trollip, Stanley R.
Computer-Based Instruction, Methods and Development, Englewood Cliffs (NJ), 1985
- Ally, Mohamed
A Team Approach to Computer Courseware Design, in: Educational Technology, Vol. 25 (1985), No. 7, pp. 28 - 30
- Anderson, Molly
Instructional Software Development: Procedures And Resources, Chapel Hill (Univ. of North Carolina), June 1986
- Aronson, Dennis T, Briggs, Leslie J.
Contributions of Gagné and Briggs to a Prescriptive Model of Instruction, in: C. M. Reigeluth (Ed.), Instructional-Design Theories and Models, An Overview of their Current Status, Hillsdale, 1983, pp. 80 - 100
- Birolini, Alessandro
Die Qualität von Software, in: Computerworld Schweiz, 1986, No. 17, pp. 7 - 12
- Bork, Alfred
Computer Futures for Education, in: Creative Computing, November 1984, pp. 178 - 180
- Bork, Alfred
Personal Computers for Education, New York, 1985
- Bork, Alfred
Pedagogical Development of CBL Material, in: A. Bork and H. Weinstock (Eds.), Designing CBL Materials, NATO ASI Series, Vol. F 23, Berlin/Heidelberg, 1986, pp. 67 - 94
- Bork, Alfred
Production Systems for CBL, in: Walker and Hess (Eds.), Instructional Software, Principles and Perspectives for Design and Use, Belmont (CA), 1984, pp. 96 - 115

- CDP/UNC-CH
Descriptions of Courseware Projects, CDP, (University of North Carolina) Chapel Hill, February 1987
- Clarke, Fayette Yvonne
An Identification of the Principles and Processes That Are Applicable in the Systematic Design of Instructional Courseware for the Microcomputer, Diss. (University of Houston), Houston-University Park, 1984, [UMI No. 85-09005]
- Collis, Betty, and Gore, Marilyn
The Collaborative Model for Instructional Software Development, in: Educational Technology, February, 1987, pp. 40 - 44
- Dixon, Terry
A Suggested Model for Development of Computer Assisted Instruction for Higher Education, ERIC Document No. 250 982, 1984
- ETC/UCI
Statement of Capabilities, Irvine, March 1987 (Handout)
- Evans, Michel W., Marciniak, John J.
Software Quality Assurance & Management, New York, 1987
- Garton, Roland, Reed, Mary Jac, Reed, George, and Stevens, Evelyne
Developing a CBI Course: The Process, in: Proceedings of the 25th International ADCIS Conference, pp. 142 - 148
- Grobe, Susan J.
Nursing Process Algorithm Used for a Nursing CAI Authoring System, in: Medinfo-83 Proceedings, J. Van Bommel and M. Ball (Eds.), Amsterdam, 1983, pp. 194 - 197
- Gupta, Sushil, Taube, Larry R.
A State of the Art Survey of Research on Project Management (1976 - 1983), in: Burton V. Dean (Ed.), Project Management: Methods and Studies, Amsterdam/New York/Oxford, 1985, pp. 293 - 313
- Hannum, W. H., & Briggs, L. J.
How Does Instructional Systems Design Differ from Traditional Instruction? in: Educational Technology, Vol. 22 (1982), No. 1, pp. 9 - 14
- Hazen, Margret
Criteria for Choosing Among Instructional Software Authoring Tools, University of North Carolina at Chapel Hill

- Heines, Jesse M. Anybody Can't Do CBT, in: Training News, 1985, March, p. 9
- Heines, Jesse M. Authoring Productivity Revised, in: Training News, Vol. 8 (1986/87), No. 7 (March), pp. 8 - 9
- Hesse, Wolfgang Methoden und Werkzeuge zur Software-Entwicklung - Ein Marsch durch die Technologie-Landschaft, in: Informatik-Spektrum, Vol. 4. (1981), pp. 229 - 245
- Hruschka, P. Integrated System Engineering Environments, in: Elektronische Rechenanlagen, Vol. 27 (1985), No. 2, pp. 60 - 68
- Hughes, B. J., Nelson, C. C., Vizo, R. Technical Management of the Development of Computer-Assisted Instruction for Complex Content, in: Performance and Instruction Journal, 1986, August, pp. 19 - 24
- IRIS/Stanford About Computing at Stanford, A Guide for Faculty and Students, Stanford University, September 1986
- Judd, Wallace Creating Lessons with Shell AS (Session 605), 5th CBT Conference, Atlanta, March 18 - 23, 1987
- Landa, Lev N. Descriptive and Prescriptive Theories of Learning and Instruction: An Analysis of Their Relationships and Interactions, in: C. M. Reigeluth (Ed.), Instructional-Design Theories and Models: An Overview of their Current Status, Hillsdale, 1983, pp. 55 - 69
- Marsh, Merle CAL in the Social Studies: Development and Evaluation of a Series of Computer Programs for Middle School Students, Diss. (University of Maryland), 1984 [UMI No. 85-12193]
- Marshall Lippincott, Jenifer Charting the CBT Learning Curve, The Six Stages of Program Development, in: Training News, Vol. 8, No. 7, pp. 11 - 13
- Maker, Steve The Courseware Group at Dartmouth College, in: SIGCUE Outlook: Computing in Higher Education, Vol. 19 (1986), Nos. 1/2, pp. 20 - 24

- Marsh, Merle CAL in the Social Studies: Development and Evaluation of a Series of Computer Programs for Middle School Students, Diss. (University of Maryland), 1984 [UMI No. 85-12193]
- Merrill, M. David Prescriptions for an Authoring System, in: JCBI, Vol. 14 (1987), No. 1 (Winter), pp. 1 - 10
- Mitchell, Mikey Leon A Minimum System for Development of Computer Assisted Instruction in Chemistry, Diss. (Kansas State University), Manhattan (Kans.), 1982 [UMI No. 83-04338]
- Montague, William E., et. al. Quality CBI Depends on Quality Instructional Design and Quality Implementation, in: Journal of Computer Based Instruction, Vol. 10 (1983), Nos. 3 & 4, pp. 90 - 93
- Osgood, Donna The Difference in Higher Education, in: Byte, 2/1987, pp. 165 - 178
- PROJECT QUEST Progress Report PROJECT QUEST, University of Texas, Austin, December, 1986
- PROJECT QUEST Progress Report PROJECT QUEST, University of Texas, Austin, January, 1987
- Reed, Mary Jac, et. al. Effective Management of CAI Projects, in: Proceedings of 26th ADCIS Conference, pp. 18 - 23
- Reigeluth, Charles M. Instructional Design: What Is It And Why Is It?, in: C. M. Reigeluth (Ed.), Instructional-Design Theories and Models, An Overview of their Current Status, Hillsdale, 1983, p. 3 - 36
- Rockart, J. F., Scott Morton, M. S. Computers and the Learning Process in Higher Education, New York, 1975, pp. 239 - 240
- Romiszowski, Alexander Joseph Designing Instructional Systems, Decision making in course planning and curriculum design, 2nd Edition (Paperback), 1986
- Sherwood, Bruce Arne Workstations at Carnegie Mellon, IEEE Computer Society Reprint, November 1986
- Shneiderman, Ben Designing the User Interface, Strategies for Effective Human-Computer Interaction, Reading (MA), 1986

- Snelbecker, Glenn E. Is Instructional Theory Alive and Well?, in: C. M. Reigeluth (Ed.), Instructional-Design Theories and Models: An Overview of their Current Status, Hillsdale, 1983, pp. 437 - 472
- Tarp, John, Russell A Comparison of Instructional Design Characteristics of CAI Lessons Produced by a Computer-based AS and by Traditional Design and Basic Programming, Diss. (University of Texas), Austin, 1982 [UMI No. 82-27729]
- Taylor, Robin Linda Mary An Analysis of Development and Design Models for Microcomputer-Based Instruction, Diss. (Syracuse University), 1986
- Thayer, Richard H., Pyster, A. B., and Wood, Roger C. Major Issues in Software Engineering Project Management, in: IEEE Transactions on Software Engineering, Vol. SE-7 (1981), No. 4, pp. 333 - 342
- Tobiasch, Rudolf Tendenzen in der Softwareentwicklung, in: Technische Rundschau, 1986, No. 52, pp. 50 - 57
- Trowbridge, David, Larkin, Jill, and Scheftic, Carol Computer-Based Tutor on Graphic Equations, CDEC, Pittsburgh (PA), 1987
- Young, Jerry L. The Case for Using Authoring Systems to Develop Courseware, in: Educational Technology, Vol. 24 (1984), No. 10, pp. 26 - 28